

# **μC/GUI**

**Graphical User Interface  
with Graphic Library**

**Version 3.26**

**Manual Rev. 0**

**Micrium Technologies Corporation**

[www.micrium.com](http://www.micrium.com)

**Empowering Embedded Systems**

## Disclaimer

Specifications written in this manual are believed to be accurate, but are not guaranteed to be entirely free of error. Specifications in this manual may be changed for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, Micrium Technologies Corporation (the distributor) assumes no responsibility for any errors or omissions and makes no warranties. The distributor specifically disclaims any implied warranty of fitness for a particular purpose.

## Copyright notice

The latest version of this manual is available as PDF file in the download area of our website at [www.micrium.com](http://www.micrium.com). You are welcome to copy and distribute the file as well as the printed version. You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of Micrium Technologie Corporation. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2002 Micrium Inc., Weston, Florida 33327-1848, U.S.A.

## Trademarks

Names mentioned in this manual may be trademarks of their respective companies. Brand and product names are trademarks or registered trademarks of their respective holders.

## Registration

Please register the software via email. This way we can make sure you will receive updates or notifications of updates as soon as they become available. For registration please provide the following information:

- Your full name and the name of your supervisor
- Your company name
- Your job title
- Your email address and telephone number
- Company name and address
- Your company's main phone number
- Your company's web site address
- Name and version of the product

Please send this information to: [licensing@micrium.com](mailto:licensing@micrium.com)

## Contact address

Micrium Inc.  
949 Crestview Circle  
Weston, FL 33327-1848  
U.S.A.  
Phone : +1 954 217 2036  
FAX : +1 954 217 2037

WEB : [www.micrium.com](http://www.micrium.com)  
Email: [support@micrium.com](mailto:support@micrium.com)

## Manual versions

This manual describes the software version 3.26. If any error occurs, please inform us and we will try to help you as soon as possible.

For further information on topics or routines not yet specified, please contact us.  
 Print date: 8/28/02

Manual version	Date	By	Explanation
3.26R0	020820	KG	Chapter 18 (Touch-Screen Support) changed to Input Devices; mouse and keyboard support added; chapter restructured. Slight modifications to section 13.10 (Scroll bar widget).
3.24R4	020809	KG	Additional macros added to Chapter 18 (Touch-Screen Support). Chapter 3 (Simulator) modified; addition of use of simulator with trial version of $\mu$ C/GUI. Section 1.7 (Data types) revised.
3.24R3	020802	KG	Additional macros added to sections 22.8 (LCDMem) and 22.9 (LCDMemC); same macros added to Chapter 20 (Low-Level Configuration).
3.24R2	020801	KG	Section 2.3 (Creating a library) revised, table and diagram added.
3.24R1	020730	KG	Minor changes throughout, including addition of () brackets to all API functions.
3.24R0	020726	KG	Chapter 9 (Colors) revised; modes 1, 2, and 444 added. Chapter 11 (Execution Model: Single Task/Multitask) added. Chapter 1 (Introduction to $\mu$ C/GUI) revised. Chapter 2 (Getting Started) revised. Chapter 18 (Time-Related Functions) changed to Timing and Execution-Related Functions; GUI_Exec() and GUI_Exec1() added. Small formatting changes throughout.
3.22R1	020723 020719	KG RS	Chapter 18 ( $\mu$ C/GUI in Multitasking Environments) merged with Chapter 22 (High-Level Configuration). Chapter 4 (Tutorial) merged with Chapter 2 (Getting Started). Chapter 10 (Colors) revised, color mode table added. GUI_X_ explanations added. Widget description enhanced, screen shots added.
3.22R0	020716	KG	Chapter 13 (Window Objects) revised; SCROLLBAR, SLIDER, RADIO and TEXT widgets added.
3.20R0	020627 020620 020618	JE KG KG	Chapter 2 (Getting Started) revised. Chapter 14 (Dialogs) revised. Chapter 13 (Window Objects) revised; CHECKBOX widgets added. Chapter 14 (Dialogs) added.
3.14R3	020618 020611 020531 020524 020507	KG KG KG KG KG	Chapter 3 (Simulator) revised. Chapter 20 (Low-Level Configuration) revised. Chapter 22 (LCD Drivers) revised. Chapter 12 (The Window Manager) revised. Version control table added.
3.14R2	020503	KG	Chapter 11 (Memory Devices) revised. Chapter 14 (Antialiasing) revised. Chapter 9 (Bitmap Converter) revised.
3.14R1	020405	KG	Completely revised for language/grammar. 1.5 (Typographic conventions) updated. Chapter 8 changed to 7.6 (Font converter). Index revised.

## Software versions

Software version	Date	By	Explanation
3.26	020820	RS	Mouse and keyboard support added.
3.24	020726	RS	GUI_Exec() and GUI_Exec1() added.
3.22	020719	RS	Support for 444 color mode added Scrollbars, Radio buttons added.
3.20	020618	RS	Dialog boxes added. Slider added. Check box added. 3D effects added.

# Table of Contents

1	Introduction to $\mu$ C/GUI .....	9
1.1	$\mu$ C/GUI .....	9
1.2	Purpose of this document .....	9
1.3	Assumptions .....	9
1.4	How to use this manual .....	10
1.5	Typographic conventions for syntax .....	10
1.6	Features.....	10
1.7	Samples and demos.....	12
1.8	Screen and coordinates .....	12
1.9	How to connect the LCD to the microcontroller .....	12
1.10	Data types.....	13
1.11	Types of configuration macros .....	14
2	Getting Started.....	15
2.1	Recommended directory structure.....	16
2.2	Configuring $\mu$ C/GUI.....	17
2.3	Using $\mu$ C/GUI with your target system.....	17
2.4	C files to include in the project .....	17
2.5	Creating a library.....	18
3	Simulator.....	19
3.1	Understanding the simulator .....	20
3.2	Using the simulator.....	21
3.3	The viewer.....	21
3.4	Device simulation and other advanced features .....	22
3.5	API reference: simulator.....	24
4	Tutorial: Using $\mu$ C/GUI with Target Hardware .....	29
4.1	Basics .....	29
4.2	How to use $\mu$ C/GUI .....	30
4.3	The "Hello world" sample program .....	30
4.4	Adding functionality to the "Hello world" program .....	31
5	Displaying Text .....	33
5.1	API reference: text .....	33
5.2	Basic routines .....	34
5.3	Routines to display text.....	34
5.4	Selecting text drawing modes.....	38
5.5	Selecting text alignment.....	40
5.6	Setting the current text position .....	42
5.7	Retrieving the current text position .....	43
5.8	Routines to clear a window or parts of it .....	43

6	Displaying Values .....	45
6.1	API reference: values .....	46
6.2	Displaying decimal values.....	46
6.3	Displaying floating-point values.....	50
6.4	Displaying binary values.....	54
6.5	Displaying hexadecimal values .....	55
7	Fonts .....	57
7.1	API reference: fonts .....	58
7.2	Selection of a font .....	58
7.3	Font-related functions.....	59
7.4	Character sets .....	62
7.5	Adding fonts.....	65
7.6	µC-FontConvert.exe font converter/editor.....	65
8	2-D Graphic Library.....	67
8.1	API reference: graphics.....	68
8.2	Drawing modes.....	69
8.3	Basic drawing routines .....	70
8.4	Drawing bitmaps.....	72
8.5	Drawing lines.....	74
8.6	Drawing polygons.....	76
8.7	Drawing circles .....	81
8.8	Drawing ellipses.....	83
8.9	Drawing arcs .....	84
9	Bitmap Converter .....	87
9.1	Introduction .....	88
9.2	Supported input formats .....	88
9.3	Supported output formats .....	88
9.4	Generating C files from bitmaps .....	88
9.5	Color conversion .....	90
9.6	Compressed bitmaps .....	92
9.7	Using a custom palette .....	92
9.8	Command line usage .....	93
9.9	Example of a converted bitmap .....	94
10	Colors.....	99
10.1	API reference: colors .....	100
10.2	Predefined colors.....	100
10.3	The color bar test routine .....	101
10.4	Fixed palette modes on color displays .....	102
10.5	Custom palette modes .....	105
10.6	Modifying the color lookup table at run-time .....	106
10.7	Basic color functions.....	107
10.8	Index/color conversion functions .....	109
10.9	Lookup table (LUT) group.....	110
11	Memory Devices .....	111
11.1	Using memory devices: an illustration.....	112
11.2	Basic functions.....	113
11.3	In order to be able to use memory devices... ..	113

11.4	API reference: memory devices .....	113
11.5	Advanced features .....	116
11.6	Banding memory device .....	120
11.7	Auto device object .....	122
12	The Window Manager (WM) .....	129
12.1	Explanation of terms .....	130
12.2	API reference: WM .....	131
12.3	Callback mechanism of the window manager .....	132
12.4	Using callback routines .....	133
12.5	Basic functions .....	134
12.6	Advanced functions .....	141
12.7	Memory device support (optional) .....	147
12.8	Example .....	147
13	Window Objects (Widgets) .....	151
13.1	Some basics .....	152
13.2	BUTTON: Button widget .....	153
13.3	Edit: Edit widget .....	162
13.4	FRAMEWIN: Frame window widget .....	168
13.5	LISTBOX: List box widget .....	174
13.6	MESSAGEBOX: Message box widget .....	183
13.7	PROGBAR: Progress bar widget .....	186
14	Dialogs .....	195
14.1	Dialog basics .....	196
14.2	Creating a dialog .....	196
14.3	API reference: dialogs .....	200
14.4	Dialog boxes .....	200
14.5	Message boxes .....	202
15	Antialiasing .....	205
15.1	Introduction .....	206
15.2	API reference: antialiasing .....	209
15.3	Control functions .....	209
15.4	Drawing functions .....	210
15.5	Examples .....	214
16	Shift-JIS Support .....	219
16.1	Displaying Shift-JIS strings .....	219
16.2	Creating Shift-JIS fonts .....	219
16.3	Example .....	219
17	Unicode .....	223
17.1	Unicode and double-byte conversions .....	224
17.2	Example .....	225
18	$\mu$ C/GUI in Multitask Environments .....	229
18.1	Multiple tasks accessing the display simultaneously .....	230
18.2	Configuration macros .....	230
18.3	Kernel interface routines .....	231
18.4	Example .....	233

19	Touch-Screen Support.....	237
19.1	Routines to be adapted .....	238
19.2	API reference: touch-screen .....	238
19.3	Example .....	240
20	Time-Related Functions .....	243
21	Low-Level Configuration .....	245
21.1	Configuring the LCD layer.....	246
21.2	General configuration .....	248
21.3	Application-specific Init and Reset .....	250
21.4	Full bus-interface configuration .....	251
21.5	Simple bus-interface configuration.....	255
21.6	LCD controller configuration: COM/SEG lines.....	258
21.7	Switches to activate routines .....	262
21.8	LCD configuration samples .....	262
22	High-Level Configuration .....	263
22.1	GUI configuration .....	264
22.2	API reference: hardware-dependent functions .....	265
23	LCD Drivers .....	269
23.1	Supported LCD controllers and respective drivers .....	270
23.2	LCD07X1.....	271
23.3	LCD13XX .....	273
23.4	LCD159A.....	275
23.5	LCD15E05 .....	276
23.6	LCD15XX .....	278
23.7	LCD6642X.....	280
23.8	LCDMem .....	281
23.9	LCDMemC .....	282
23.10	LCDSLin.....	284
24	LCD Driver API .....	285
24.1	API reference: $\mu$ C/GUI LCD.....	286
24.2	Init & display control group .....	287
24.3	Drawing group .....	288
24.4	"Get" group .....	290
24.5	Lookup table (LUT) group.....	291
24.6	Miscellaneous group .....	291
25	Performance and Resource Usage.....	295
25.1	Performance benchmark .....	296
25.2	Memory requirements.....	297
26	Standard Fonts .....	299
26.1	Font identifier naming convention.....	299
26.2	Font file naming convention .....	300
26.3	Measurement, ROM-size and character set of fonts .....	301
26.4	Proportional fonts.....	301
26.5	Monospaced fonts .....	318

27	Questions and Answers .....	327
----	-----------------------------	-----



# Chapter 1

## Introduction to $\mu$ C/GUI

---

### $\mu$ C/GUI

$\mu$ C/GUI is a Universal graphic software for embedded applications. It is designed to provide an efficient, processor and LCD-controller independent graphical user interface to any application using a graphical LCD. It works in single as well as in multi-task environments.  $\mu$ C/GUI can be adopted to any size of physical and virtual display with any LCD-controller and CPU.

Its design is modular, consisting of different layers in different modules. One layer - called the LCD-driver - covers all access to the LCD.  $\mu$ C/GUI works with all CPUs, since it is written in 100% ANSI-"C".

$\mu$ C/GUI is well-suited for most applications using black/ white and color LCDs. It has a very good color management which allows it to handle grayscales.  $\mu$ C/GUI also features an extensive 2-D graphic library and a window manager which supports windows while using a minimum of RAM.

### Purpose of this document

This guide describes how to install, configure and use the  $\mu$ C/GUI graphical user interface for embedded applications. It also explains the internal structure of the software.

### Assumptions

This guide assumes that you already possess a solid knowledge of the C programming language.

If you feel that your knowledge of C is not sufficient, we recommend *The C Programming Language* by Kernighan and Richie, which describes the programming standard and, in newer editions, also covers the ANSI C standard. Knowledge of assembly programming is not required.

## 1.1 Requirements

A target system is not required in order to develop software with  $\mu$ C/GUI; most of the software can be developed using the simulator. However, the final purpose is usually to be able to run the software on a target system.

### Target system (hardware)

Your target system must:

- Have a CPU (8/16/32/64 bits)
- Have a minimum of RAM and ROM
- Have a full graphic LCD (any type and any resolution)

The memory requirements vary depending on which parts of the software are used and how efficient your target compiler is. It is therefore not possible to specify precise values, but the following apply to typical systems.

#### Small systems (no window manager)

- RAM: 100 bytes
- Stack: 500 bytes
- ROM: 10-25 kb (depending on the functionality used)

#### Big systems (including window manager and widgets)

- RAM: 2-6 kb (depending on number of windows required)
- Stack: 1200 bytes
- ROM: 30-60 kb (depending on the functionality used)

Note that ROM requirements will increase if your application uses many fonts. All values are rough estimates and cannot be guaranteed.

### Development environment (compiler)

The CPU used is of no importance; only an ANSI-compliant "C" compiler is required. If your compiler has some limitations, please let us know and we will inform you if these will be a problem when compiling the software. Any compiler for 16/32/64-bit CPUs or DSPs that we know of can be used; most 8-bit compilers can be used as well.

A C++ compiler is not required, but can be used. The application program can therefore also be programmed in C++ if desired.

## 1.2 $\mu$ C/GUI $\mu$ C/GUIatures

$\mu$ C/GUI is designed to provide an efficient, processor- and LCD controller-independent graphical user interface for any application that operates with a graphical LCD. It is compatible with single-task and multitask environments, with a proprietary operating system or with any commercial RTOS.  $\mu$ C/GUI is shipped as "C" source code. It may be adapted to any size physical and virtual display with any LCD controller and CPU. Its features include the following:

#### General

- Any 8/16/32-bit CPU; only an ANSI "C" compiler is required.
- Any (monochrome, grayscale or color) LCD with any controller supported (if the right driver is available).
- May work without LCD controller on smaller displays.
- Any interface supported using configuration macros.

- Display-size configurable.
- Characters and bitmaps may be written at any point on the LCD, not just on even-numbered byte addresses.
- Routines are optimized for both size and speed.
- Compile time switches allow for different optimizations.
- For slower LCD controllers, LCD can be cached in memory, reducing access to a minimum and resulting in very high speed.
- Clear structure.
- Virtual display support; the virtual display can be larger than the actual display.

### **Graphic library**

- Bitmaps of different color depths supported.
- Bitmap converter available.
- Absolutely no floating-point usage.
- Fast line/point drawing (without floating-point usage).
- Very fast drawing of circles/polygons.
- Different drawing modes.

### **Fonts**

- uA variety of different fonts are shipped with the basic software: 4\*6, 6\*8, 6\*9, 8\*8, 8\*9, 8\*16, 8\*17, 8\*18, 24\*32, and proportional fonts with pixel-heights of 8, 10, 13, 16. For more information, see Chapter 25: "Standard Fonts".
- New fonts can be defined and simply linked in.
- Only the fonts used by the application are actually linked to the resulting executable, resulting in minimum ROM usage.
- Fonts are fully scalable, separately in X and Y.
- Font converter available; any font available on your host system (i.e. Microsoft Windows) can be converted.

### **String/value output routines**

- Routines to show values in decimal, binary, hexadecimal, any font.
- Routines to edit values in decimal, binary, hexadecimal, any font.

### **Window manager (WM)**

- Complete window management including clipping. Overwriting of areas outside a window's client area is impossible.
- Windows can be moved and resized.
- Callback routines supported (usage optional).
- WM uses minimum RAM (app. 20 bytes per window).

### **Optional widgets for PC look and feel**

- Widgets (window objects, also known as controls) are available. They generally operate automatically and are simple to use.

### **Touch-screen & mouse support**

- For window objects such as the button widget,  $\mu$ C/GUI offers touch-screen and mouse support.

### **PC tools**

- Simulation plus viewer.
- Bitmap converter.
- Font converter.u

## Samples and demos

To give you a better idea of what  $\mu$ C/GUI can do, we have different demos available as "ready-to-use" simulation executables under `Sample\EXE`. The source of the sample programs is located in the folder `Sample`. The folder `Sample\GUIDemo` contains an application program showing most of  $\mu$ C/GUI's features.

## 1.3 Evaluation board

A complete Evaluation board including a demoboard with LCD, a "C" compiler and a sample project are available. It has been designed primarily to test and demonstrate  $\mu$ C/GUI, and it can be used to get familiar with the software.



### Evaluation board

- Evaluation board with Mitsubishi M30803 CPU and SED 13705 LCD controller (including schematic and documentation).
- LCD (320\*240 pixel) either monochrome, 1/4 VGA color display or TFT.

For more details, please take a look at our website at [www.micrium.com](http://www.micrium.com).

## 1.4 How to use this manual

This manual explains how to install, configure and use  $\mu$ C/GUI. It describes the internal structure of the software and all the functions that  $\mu$ C/GUI offers (the Application Program Interface, or API).

Before actually using  $\mu$ C/GUI, you should read or at least glance through this manual in order to become familiar with the software. The following steps are then recommended:

- Copy the  $\mu$ C/GUI files to your computer.
- Go through Chapter 2: "Getting Started".
- Use the simulator in order to become more familiar with what the software can do (refer to Chapter 3: "Simulator").
- Expand your program using the rest of the manual for reference.

## Typographic conventions for syntax

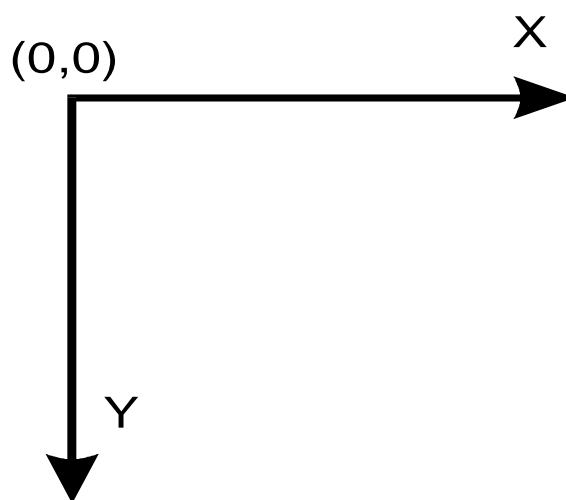
This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
Keyword	Text that you enter at the command-prompt or that appears on the display (i.e. system functions, file- or pathnames).
Parameter	Parameters in API functions.
Sample	Sample code in program examples.
New Sample	Sample code that has been added to an existing program example.

## 1.5 Screen and coordinates

The screen consists of many dots that can be controlled individually. These dots are called pixels. Most of the text and drawing functions that  $\mu$ C/GUI offers in its API to the user program can write or draw on any specified pixel.

The horizontal scale is called the X-axis, whereas the vertical scale is called the Y-axis. Coordinates are denoted as a pair consisting of an X- and a Y-value (X, Y). The X-coordinate is always first in routines that require X and Y coordinates. The upper left corner of the display (or a window) has per default the coordinates (0,0). Positive X-values are always to the right; positive Y-values are always down. The above graph illustrates the coordinate system and directions of the X- and Y- axes. All coordinates passed to an API function are always specified in pixels.



## 1.6 How to connect the LCD to the microcontroller

$\mu$ C/GUI handles all access to the LCD. Virtually any LCD controller can be supported, independently of how it is accessed. For details, please refer to Chapter 20: "Low-Level Configuration". Also, please get in contact with us if your LCD controller is not supported. We are currently writing drivers for all LCD controllers available on the market and may already have a proven driver for the LCD controller that you intend to use. It is usually very simple to write the routines (or macros) used to access the LCD in your application. Micrium offers the customization service, if necessary with your target hardware.

It does not really matter how the LCD is connected to the system as long as it is somehow accessible by software, which may be accomplished in a variety of ways. Most of these interfaces are supported by a driver which is supplied in source code form. This driver does not normally require modifications, but is configured for your hardware by making changes in the file `LCDConf.h`. Details about how to customize a

driver to your hardware as necessary are explained in Chapter 22: "LCD Drivers". The most common ways to access the LCD are described as follows. If you simply want to understand how to use  $\mu$ C/GUI, you may skip this section.

### **LCD with memory-mapped LCD controller:**

The LCD controller is connected directly to the data bus of the system, which means the controller can be accessed just like a RAM. This is a very efficient way of accessing the LCD controller and is most recommended. The LCD addresses are defined to the segment LCDSEG, and in order to be able to access the LCD the linker/locator simply needs to be told where to locate this segment. The location must be identical to the access address in physical address space. Drivers are available for this type of interface and for different LCD controllers.

### **LCD with LCD controller connected to port / buffer**

For slower LCD controllers used on fast processors, the use of port-lines may be the only solution. This method of accessing the LCD has the disadvantage of being somewhat slower than direct bus-interface but, particularly with a cache that minimizes the accesses to the LCD, the LCD update is not slowed down significantly. All that needs to be done is to define routines or macros which set or read the hardware ports/buffers that the LCD is connected to. This type of interface is also supported by different drivers for the different LCD controllers.

### **Proprietary solutions: LCD without LCD controller**

The LCD can also be connected without an LCD controller. In this case, the LCD data is usually supplied directly by the controller via a 4- or 8-bit shift register. These proprietary hardware solutions have the advantage of being inexpensive, but the disadvantage of using up much of the available computation time. Depending on the CPU, this can be anything between 20 and almost 100 percent; with slower CPUs, it is really not possible at all. This type of interface does not require a specific LCD driver because  $\mu$ C/GUI simply places all the display data into the LCD cache. You must write the hardware-dependent portion that periodically transfers the data in the cache memory to your LCD.

Sample code for transferring the video image into the display is available in both "C" and optimized assembler for M16C and M16C/80.

## 1.7 uData types

Since "C" does not provide data types of fixed lengths which are identical on all platforms,  $\mu$ C/GUI uses, in most cases, its own data types as shown in the table below:

Data type	Definition	Explanation
I8	signed char	8-bit signed value
U8	unsigned char	16-bit unsigned value
I16	signed short	16-bit signed value
U16	unsigned short	16-bit unsigned value
I32	signed long	32-bit signed value
U32	unsigned long	32-bit unsigned value
I16P	signed short	16-bit (or more) signed value
U16P	unsigned short	16-bit (or more) unsigned value

For most 16/32-bit controllers, the settings will work fine. However, if you have similar defines in other sections of your program, you might want to change or relocate them. A recommended place is in the configuration file `LCDConf.h`.



# Chapter 2

## Getting Started

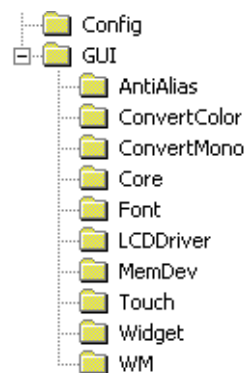
---

The following chapter provides an overview of the basic procedures for setting up and configuring  $\mu$ C/GUI on your target system. It also includes a simple program example.

Please keep in mind that most topics are treated in greater detail in later chapters. You will most likely need to refer to other parts of the manual before you begin more complex programming.

## 2.1 Recommended directory structure

We recommend keeping  $\mu$ C/GUI separate from your application files. It is good practice to keep all the program files (including the header files) together in the GUI subdirectories of your project's root directory. The directory structure should be similar to the one pictured on the right. This practice has the advantage of being very easy to update to newer versions of  $\mu$ C/GUI by simply replacing the GUI\ directories. Your application files can be stored anywhere.



### Subdirectories

The following table shows the contents of all GUI subdirectories:

Directory	Contents
Config	Configuration files
GUI\AntiAlias	Antialiasing support *
GUI\ConvertMono	Color conversion routines used for b/w and grayscale displays
GUI\ConvertColor	Color conversion routines used for color displays
GUI\Core	$\mu$ C/GUI core files
GUI\Font	Font files
GUI\LCDDriver	LCD driver
GUI\MemDev	Memory device support *
GUI\Touch	Touch-panel support *
GUI\Widget	Widget library *
GUI\WM	Window manager *

(\* = optional)

### Include directories

You should make sure that the include path contains the following directories (the order of inclusion is of no importance):

- Config
- GUI\Core
- GUI\Widget (if using widget library)
- GUI\WM (if using window manager)

### Warning: Always make sure that you have only one version of each file!

It is frequently a major problem when  $\mu$ C/GUI updating to a new version of  $\mu$ C/GUI if you have old files included and therefore mix different versions. If you keep  $\mu$ C/GUI in the directories as suggested (and only in these), this type of problem cannot occur. When updating to a newer version, you should be able to keep your configuration files and leave them unchanged. For safety reasons, we recommend backing (or at least renaming) the GUI\ directories prior to updating.

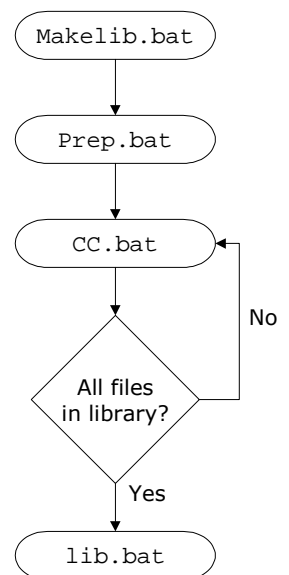
## 2.2 Adding $\mu$ C/GUI to the target program

You basically have a choice between including only the source files that you are actually going to use in your project, which will then be compiled and linked, or creating a library and linking the library file. If your tool chain supports "smart" linking (linking in only the modules that are referenced and not those that are unreferenced), there is no real need to create a library at all, since only the functions and data structures which are required will be linked. If your tool chain does not support "smart" linking, a library makes sense, because otherwise everything will be linked in and the program size will be excessively large. For some CPUs, we have sample projects available to help you get started.

## 2.3 Creating a library

Building a library from the sources is a simple procedure. The first step is to copy the batch files (located under `Sample\Makelib`) into your root directory. Then, make any necessary changes. There are a total of four batch files which need to be copied, described in the table below. The main file, `Makelib.bat`, will be the same for all systems and requires no changes. To build a library for your target system, you will normally need to make slight modifications to the other three smaller files. Finally, start the file `Makelib.bat` to create the library. The batch files assume that your GUI and Config subdirectories are set up as recommended.

The procedure for creating a library is illustrated in the flow chart to the right. The `Makelib.bat` file first calls `Prep.bat` to prepare the environment for the tool chain. Then it calls `CC.bat` for every file to be included in the library. It does this as many times as necessary. `CC.bat` adds each object file to a list that will be used by `lib.bat`. When all files to be added to the library have been listed, `Makelib.bat` then calls `lib.bat`, which uses a librarian to put the listed object files into the actual library.



File	Explanation
Makelib.bat	Main batch file. No modification required.
Prep.bat	Called by Makelib.bat to prepare environment for the tool chain to be used,
CC.bat	Called by Makelib.bat for every file to be added to the library; creates a list of these object files which will then be used in the next step by the librarian in the lib.bat file.
lib.bat	Called by Makelib.bat to put the object files listed by CC.bat into a library.

The files as shipped assume that a Microsoft compiler is installed in its default location. If all batch files are copied to the root directory (directly above GUI) and no changes are made at all, a simulation library will be generated for the  $\mu$ C/GUI simulation. In order to create a target library, however, it will be necessary to modify `Prep.bat`, `CC.bat`, and `lib.bat`.

## 2.4 $\mu$ C/GUI "C" files to include in the project

Generally speaking, you need to include the core "C" files of  $\mu$ C/GUI, the LCD driver, all font files you plan to use and any optional modules you have ordered with  $\mu$ C/GUI:

- All "C" files of the folders `GUI\Core`, `GUI\ConvertColor` and `GUI\ConvertMono`
- The fonts you plan to use (located in `GUI\Font`)

### Additional software packages

If you plan to use additional, optional modules you must also include their "C" files:

- Antialiasing: all "C" files located in `GUI\AntiAlias`
- Memory devices: all "C" files located in `GUI\MemDev`
- Touch-panel support: all "C" files located in `GUI\Touch`
- Widget library: all "C" files located in `GUI\Widget`
- Window Manager: all "C" files located in `GUI\WM`

### Using the hardware

- LCD driver: All "C" files of the folder `GUI\LCDDriver` except `LCDWin.c`.
- `GUI_X.c`. A sample file is available as `Sample\GUI_X`. This file contains the hardware-dependent part of  $\mu$ C/GUI and should be modified as described in Chapter 21: "High-Level Configuration".

### Using the simulator

- LCD driver: `GUI\LCDDriver\LCDWin.c`.

Be sure that you include `GUI.h` in all of your source files that access  $\mu$ C/GUI.

## 2.5 $\mu$ C/GUI Configuring $\mu$ C/GUI

The `Config` folder should contain the configuration files matching your order. The file `LCDConf.h` normally contains all the definitions necessary to adopt  $\mu$ C/GUI to your LCD, which is the main task when configuring  $\mu$ C/GUI. For details, please see Chapter 20: "Low-Level Configuration".

If  $\mu$ C/GUI is not configured correctly, because you did not select the right display resolution or chose the wrong LCD controller, it will probably not display anything at all or display something that does not resemble what you expected. So take care to tailor `LCDConf.h` to your needs. If you do not wish to deal with this process, Micrium can do it for you, as well as test  $\mu$ C/GUI in your application with your hardware and your LCD.

## $\mu$ C/GUI Types of configuration macros

The following types of configuration macros exist:

### Binary switches "B"

Switches can have a value of either 0 or 1, where 0 means deactivated and 1 means activated (actually anything other than 0 would work, but using 1 makes it easier to read a `config` file). These switches can enable or disable a certain functionality or behavior. Switches are the simplest form of configuration macro.

### Numerical values "N"

Numerical values are used somewhere in the code in place of a numerical constant. Typical examples are in the configuration of the resolution of an LCD.

### Selection switches "S"

Selection switches are used to select one out of multiple options where only one of those options can be selected. A typical example might be the selection of the type of LCD controller used, where the number selected denotes which source code (in which LCD driver) is used to generate object code.

### Alias "A"

A macro which operates like a simple text substitute. An example would be the define `U8`, in which the preprocessor would replace with `unsigned char`.

### Function replacements "F"

Macros can basically be treated like regular functions although certain limitations apply, as a macro is still put into the code as simple text replacement. Function replacements are mainly used to add specific functionality to a module (such as the access to an LCD) which is highly hardware-dependent. This type of macro is always declared using brackets (and optional parameters). `μC/GUI`

## 2.6 `μC/GUI` Initializing `μC/GUI`

The routine `GUI_Init()` initializes the LCD and the internal data structures of `μC/GUI`, and must be called before any other `μC/GUI` function. This is done by placing the following line into the init sequence of your program:

```
GUI_Init();
```

If this call is left out, the entire graphics system will not be initialized and will therefore not be ready.

## 2.7 Using `μC/GUI` with target hardware

The following is just a basic outline of the general steps that should be taken when starting to program with `μC/GUI`. All steps are explained further in subsequent chapters.

### Step 1: Customizing `μC/GUI`

The first step is usually to customize `μC/GUI` by modifying the header file `LCDConf.h`. You must define the basic data types (`U8`, `U16`, etc.) and mandatory configuration switches regarding resolution of the display and the LCD controller used.

### Step 2: Defining access addresses or access routines

For memory-mapped LCDs, the access addresses of the LCD simply need to be defined in `LCDConf.h`.

For port/buffer-accessed LCDs, interface routines must be defined. Samples of the required routines are available under `Samples\LCD_X` or on our website in the download area.

### Step 3: Compiling, linking and testing the sample code

`μC/GUI` comes with sample code for both single- and multitask environments. Compile, link and test these little sample programs until you feel comfortable doing so.

### Step 4: Modifying the sample program

Make simple modifications to the sample programs. Add additional commands such as displaying text in different sizes on the display, showing lines and so on.

### Step 5: $\mu$ C/GUI in multitask applications: adapt to your OS (if necessary)

If multiple tasks should be able to access the display simultaneously, the macros `GUI_MAXTASK` and `GUI_OS` come into play, as well as the file `GUI_Task.c`. For details and sample adaptations, please refer to Chapter 21: "High-Level Configuration".

### Step 6: Write your own application using $\mu$ C/GUI

By now you should have a clearer understanding of how to use  $\mu$ C/GUI. Think about how to structure the program your application requires and use  $\mu$ C/GUI by calling the appropriate routines. Consult the reference chapters later in this manual, as they discuss the specific  $\mu$ C/GUI functions and configuration macros that are available.

## 2.8 The "Hello world" sample program

A "Hello world" program has been used as a starting point for "C" programming since the early days, because it is essentially the smallest program that can be written. A "Hello world" program with  $\mu$ C/GUI, called `HELLO.c`, is shown below and is available as `BASIC>HelloWorld.c` in the sample shipped with  $\mu$ C/GUI.

The whole purpose of the program is to write "Hello world" in the upper left corner of the display. In order to be able to do this, the hardware of the application, the LCD and the GUI must first be initialized.  $\mu$ C/GUI is initialized by a call to `GUI_Init()` at the start of the program, as described previously. In this example, we assume that the hardware of your application is already initialized.

The "Hello world" program looks as follows:

```

/*****
 *                               *
 *           Micrium Inc.       *
 *      Empowering embedded systems      *
 *                               *
 *            $\mu$ C/GUI sample code      *
 *                               *
 *****/

-----
File       : BASIC>HelloWorld.c
Purpose    : Simple demo drawing "Hello world"
-----
*/

#include "GUI.H"

/*****
 *                               *
 *           main               *
 *                               *
 *****/

void main(void) {
/*
 *ToDo:  Make sure hardware is initilized first!!
 */
    GUI_Init();
    GUI_DisString("Hello world!");
    while(1);
}

```

## Adding functionality to the "Hello world" program

Our little program has not been doing too much so far. We can now extend the functionality a bit: after displaying "Hello world", we would like the program to start counting on the display in order to be able to estimate how fast outputs to the LCD can be made. We can simply add a bit of code to the loop at the end of the main program, which is essentially a call to the function that displays a value in decimal form.

The example is available as `BASIC_Hello1.c` in the sample folder.

```

/*****
*                               *
*           Micrium Inc.       *
*       Empowering embedded systems   *
*                               *
*           µC/GUI sample code       *
*                               *
*****/

-----
File      : BASIC_Hello1.c
Purpose   : Simple demo drawing "Hello world"
-----
*/

#include "GUI.H"

/*****
*                               *
*           main               *
*                               *
*****/

void main(void) {
    int i=0;
    /*
    ToDo:  Make sure hardware is initilized first!!
    */
    GUI_Init();
    GUI_DispString("Hello world!");
    while(1) {
        GUI_DispDecAt( i++, 20,20,4);
        if (i>9999) i=0;
    }
}

```



# Chapter 3

## Simulator

---

The PC simulation of  $\mu$ C/GUI allows you to compile the same "C" source on your Windows PC using a native (typically Microsoft) compiler and create an executable for your own application. Doing so allows the following:

- Design of the user interface on your PC (no need for hardware!).
- Debugging of your user interface program.
- Creation of demos of your application, which can be used to discuss the user interface.

The resulting executable can be easily sent via email.



## 3.1 Understanding the simulator

The  $\mu$ C/GUI simulator uses Microsoft Visual C++ (version 6.00 or higher) and the integrated development environment (IDE) which comes with it. You will see a simulation of your LCD on your PC screen, which will have the same resolution in X and Y and can display the exact same colors as your LCD once it has been properly configured. The entire graphic library API and window manager API of the simulation are identical to those on your target system; all functions will behave in the very same way as on the target hardware since the simulation uses the same "C" source code as the target system. The difference lies only in the lower level of the software: the LCD driver. Instead of using the actual LCD driver, the PC simulation uses a simulation driver which writes into a bitmap. The bitmap is then displayed on your screen using a second thread of the simulation. This second thread is invisible to the application; it behaves just as if the LCD routines were writing directly to the display.

## 3.2 Using the simulator in the trial $\mu$ C/GUI version

The trial version of  $\mu$ C/GUI contains a full library which allows you to evaluate all available features of  $\mu$ C/GUI. It also includes the  $\mu$ C/GUI viewer (used for debugging applications), as well as demo versions of the font converter and the bitmap converter. Keep in mind that, being a trial version, you will not be able to change any configuration settings or view the source code, but you will still be able to become familiar with what  $\mu$ C/GUI can do.

### Directory structure

The directory structure of the simulator in the trial version will appear as pictured to the right.

The `Application` directory contains the source of the demo program.

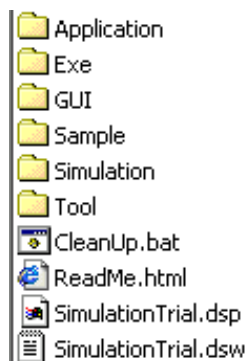
The `Exe` directory contains a ready-to-use demo program.

The `GUI` directory contains library files and include files needed to use the library.

The `Sample` directory contains simulation samples and their sources.

The `Simulation` directory contains the files needed for the simulation.

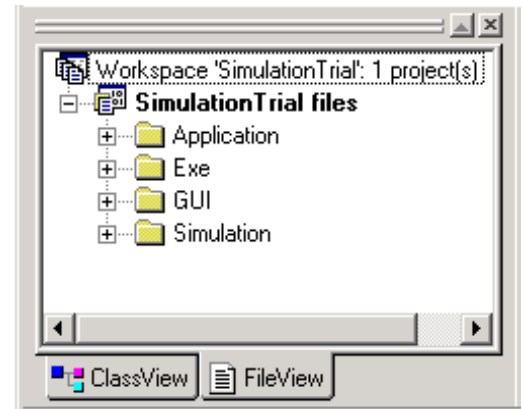
The `Tool` directory contains the  $\mu$ C/GUI viewer, a demo version of the bitmap converter and a demo version of the font converter.



## Visual C++ workspace

The root directory shown above includes the Microsoft Visual C++ workspace (Simulation-Trial.dsw) and project files (Simulation-Trial.dsp). Double-click the workspace file to open the Microsoft IDE.

The directory structure of the Visual C++ workspace will look like the one shown to the right.



## Compiling the demo program

The source files for the demo program are located in the `Application` directory as a ready-to-go simulation, meaning that you need only to rebuild and start it. Please note that to rebuild the executable, you will need to have Microsoft Visual C++ (version 6.00 or later) installed.

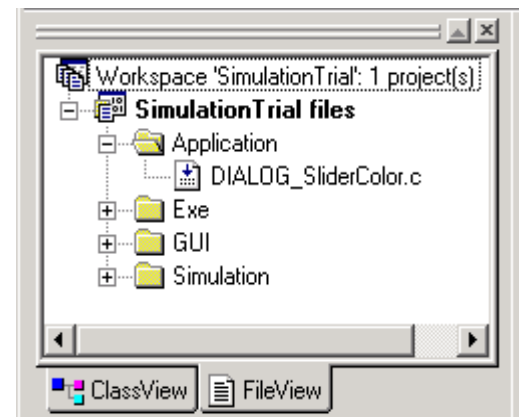
- Step 1: Open the Visual C++ workspace by double-clicking on `Simulation-Trial.dsw`.
- Step 2: Rebuild the project by choosing `Build/Rebuild All` from the menu (or by pressing F7).
- Step 3: Start the simulation by choosing `Build/Start Debug/Go` from the menu (or by pressing F5).

The demo project will begin to run and may be exited at any time by right-clicking on it and selecting `Exit`.

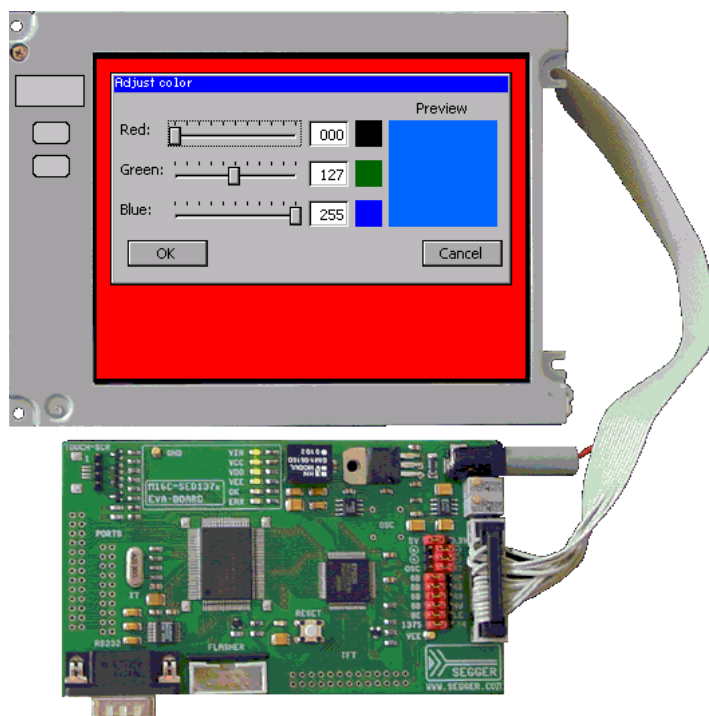
## Compiling the samples

The `Sample` directory contains ready-to-go samples that demonstrate different features of  $\mu$ C/GUI and provide examples of some of their typical uses. In order to build any of these executables, their "C" source must be added to the project. This is easily done with the following procedure:

- Step 1: Double-click on the `Application` folder in the Visual C++ workspace. The demo files will be displayed beneath it.
- Step 2: Remove all files from the `Application` folder by selecting them and pressing the Delete key. They will not be deleted; you are just removing them from the project.
- Step 3: You should now have an empty `Application` folder. Right-click on it and select `Add files to folder`. A dialog box will be displayed.
- Step 4: Double-click on the `Sample` folder, and select one of the sample files inside. Your workspace directory should now appear similar to the one on the right. Of course, the file name may be different; the important thing here is that the `Application` folder contains only the "C" file of the sample you want to compile, and nothing else.
- Step 5: Rebuild the sample by choosing `Build/Rebuild All` from the menu (or by pressing F7).



- Step 6: Start the simulation by choosing `Build/Start Debug/Go` from the menu (or by pressing F5). The result of the sample selected above is pictured below:

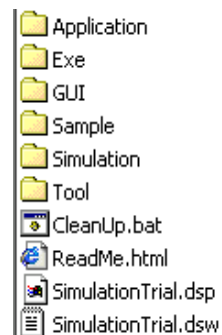


## 3.3 Using the simulator with the $\mu$ C/GUI source

### Directory structure

The root directory of the simulator can be anywhere on your PC, e.g. `C:\work\GSCSim`. The directory structure will appear as shown to the right. This structure is very similar to that which we recommend for your target application (see Chapter 2: "Getting Started" for more information). The subdirectories containing  $\mu$ C/GUI program files are in the `GUI` folder and should contain the exact same files as the directories of the same names which you are using for your target (cross) compiler. You should not make any changes to the `GUI` subdirectories, as this would make updating to a newer version of  $\mu$ C/GUI more difficult.

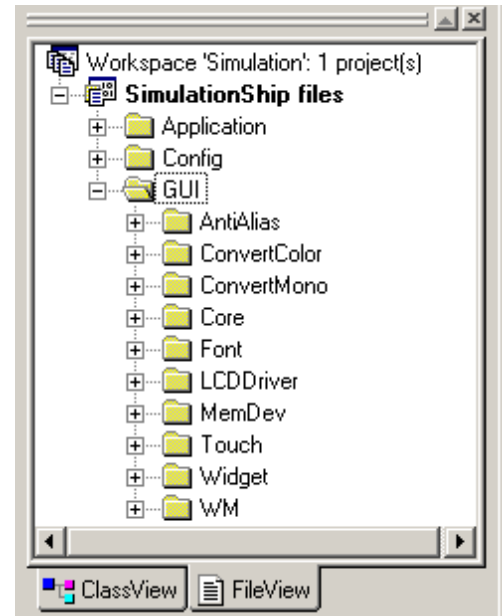
The `Config` directory contains configuration files which need to be modified in order to reflect your target hardware settings (mainly LCD-size and colors which can be displayed).



## Visual C++ workspace

The root directory shown above includes the Microsoft Visual C++ workspace (`Simulation.dsw`) and project files (`Simulation.dsp`). The workspace allows you to modify an application program and debug it before compiling it on your target system.

The directory structure of the Visual C++ workspace will appear similar to that shown to the right. Here, the `GUI` folder is open to display the `μC/GUI` subdirectories. Please note that your `GUI` directory may not look exactly like the one pictured, depending on which additional features of `μC/GUI` you have. The folders `Core`, `Font` and `LCDDriver` are part of the basic `μC/GUI` package and will always appear in the workspace directory.



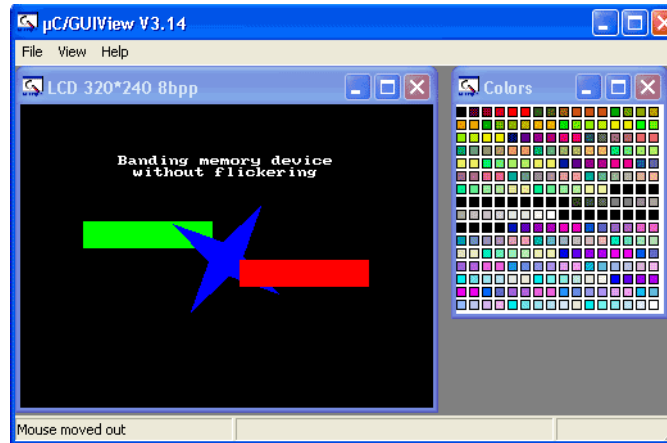
## Compiling for your application

The demo simulation contains one or more application "C" files (located in the `Application` directory), which can be modified. You may also add files to or remove files from the project. Typically you would want to at least change the bitmap to your own company logo or image of choice. You should then rebuild the program within the Visual C++ workspace in order to test/debug it. Once you have reached a point where you are satisfied with the result and want to use the program in your application, you should be able to compile these same files on your target system and get the same result on the target display. The general procedure for using the simulator would be as follows:

- Step 1: Open the Visual C++ workspace by double-clicking on `Simulation.dsw`.
- Step 2: Compile the project by choosing `Build/Rebuild All` from the menu (or by pressing F7).
- Step 3: Run the simulation by choosing `Build/Start Debug/Go` from the menu (or by pressing F5).
- Step 4: Replace the bitmap with your own logo or image.
- Step 5: Make further modifications to the application program as you wish, by editing the source code or adding/deleting files.
- Step 6: Compile and run the application program within Visual C++ to test the results. Continue to modify and debug as needed.
- Step 7: Compile and run the application program on your target system.

## 3.4 The viewer

If you use the simulator to debug your application, you cannot see the LCD output when stepping through the source code. The viewer solves this problem by showing the LCD window and the color window of your simulation. The viewer can be found under `Tool\µC-GUI-View.exe`.



### Using the simulator and the viewer

It is your choice if you want to start the viewer before debugging your application or while you are debugging. Our suggestion:

- Step 1: Start the viewer. No LCD- or color window is shown until the simulation has been started.
- Step 2: Open the Visual C++ workspace.
- Step 3: Compile and run the application program.
- Step 4: Debug the application as described previously.

The advantage is that you can now follow all drawing operations step by step in the LCD window. Per default the viewer window is always on top. You can change this behavior by selecting `View\Always on top` from the menu.

## 3.5 Device simulation and other advanced features

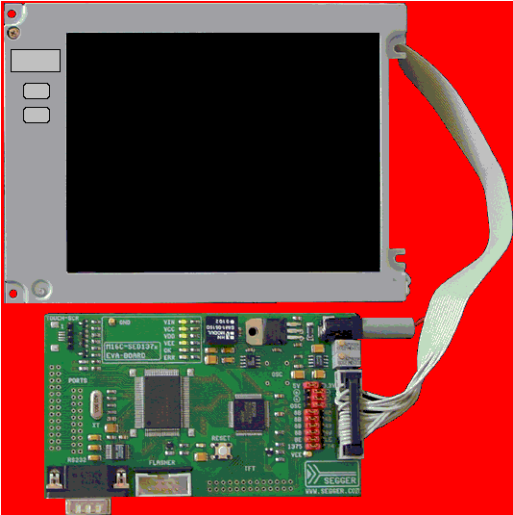

**Warning: Device simulation, and other features which are based on it, are advanced features which may require the simulator source code in order to work for your target system. This source code is not normally shipped with µC/GUI. Please contact us for more information.**

The simulator can show the simulated LCD in a bitmap of your choice, typically your target device. The bitmap can be dragged over the screen and may, in certain applications, be used to simulate the behavior of the entire target device.

In order to simulate the appearance of the device, a bitmap is required. This bitmap is usually a photo (top view) of the device, and must be named `Device.bmp`. It may be a separate file (in the same directory as the executable), or it may be included as a resource in the application by including the following line in the resource file (extension `.rc`):

145 BITMAP DISCARDABLE "Device.bmp"  
For more information, please refer to the Win32 documentation.

The size of the bitmap should be such that the size of the area in which the LCD will be shown equals the resolution of the simulated LCD. This is best seen in the following example:

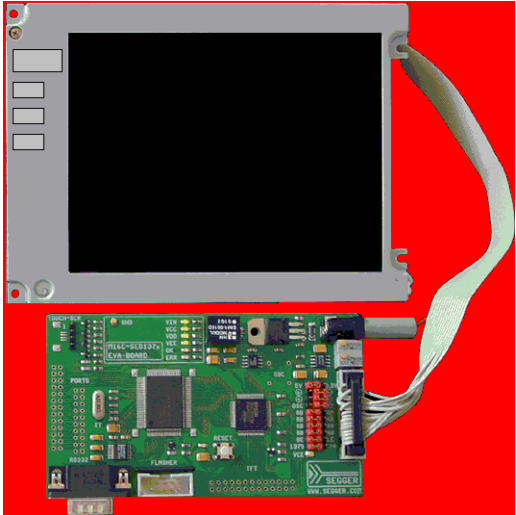
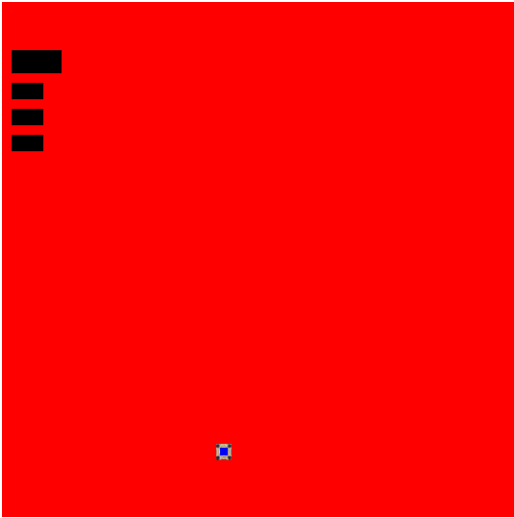
Device bitmap (Device.bmp)	Device including simulated LCD as visible on screen
	

The red area is automatically made transparent. The transparent areas do not have to be rectangular; they can have an arbitrary shape (up to a certain complexity which is limited by your operating system, but is normally sufficient). Bright red (0xFF0000) is the default color for transparent areas, mainly because it is not usually contained in most bitmaps. To use a bitmap with bright red, the default transparency color may be changed with the function `SIM_SetTransColor`.

**Hardkey simulation**

Hardkeys may also be simulated as part of the device, and may be selected with the mouse pointer. The idea is to be able to distinguish whether a key or button on the simulated device is pressed or unpressed. A hardkey is considered "pressed" as long as the mouse button is held down; releasing the mouse button or moving the pointer off of the hardkey "unpresses" the key. A toggle behavior between pressed and unpressed may also be specified with the routine `SIM_HARDKEY_SetMode`. In order to simulate hardkeys, you need a second bitmap of the device which is transparent except for the keys themselves (in their pressed state). This bitmap can again be in a seperate file in the directory, or included as a resource in the executable. The filename needs to be `Device1.bmp`, and the following lines would typically be included in the resource file (extension `.rc`):  
145 BITMAP DISCARDABLE "Device.bmp"  
146 BITMAP DISCARDABLE "Device1.bmp"

Although hardkeys may be any shape, it is very important that the two bitmaps are the same size in pixels, so that the hardkeys in `Device1.bmp` will overlay those in `Device.bmp` exactly. The following example illustrates this:

Device bitmap: unpressed hardkey state (Device.bmp)	Device hardkey bitmap: pressed hardkey state (Device1.bmp)
	

When a key is "pressed" with the mouse, the corresponding section of the hardkey bitmap (`Device1.bmp`) will overlay the device bitmap in order to display the key in its pressed state.

The keys may be polled periodically to determine if their states (pressed/unpressed) have changed and whether they need to be updated. Alternatively, a callback routine may be set to trigger a particular action to be carried out when the state of a hardkey changes.

### 3.6 Simulator API

All of the simulator API functions must be called in the setup phase. The calls should ideally be done from within the routine `SIM_X_Init()`, which is located in the file `SIM_X.c`. The example below calls `SIM_SetLCDPos()` in the setup:

```
*/
#include <windows.h>
#include <stdio.h>
#include "SIM.h"

void SIM_X_Init() {
    SIM_SetLCDPos(0,0);    // Define the position of the LCD in the bitmap
}
```

The table below lists the available simulation-related routines in alphabetical order within their respective categories. Detailed descriptions of the routines follow:

Routine	Explanation
<b>Device simulation</b>	
<a href="#">SIM_SetLCDPos()</a>	Set the position for the simulated LCD within the target device bit-map.
<a href="#">SIM_SetTransColor()</a>	Set the color to be used for transparent areas.
<b>Hardkey simulation</b>	
<a href="#">SIM_HARDKEY_GetNum()</a>	Return the number of available hardkeys.
<a href="#">SIM_HARDKEY_GetState()</a>	Return the state of a specified hardkey (0 = unpressed, 1 = pressed).
<a href="#">SIM_HARDKEY_SetCallback()</a>	Set a callback routine to be executed when the state of a specified hardkey changes.
<a href="#">SIM_HARDKEY_SetMode()</a>	Set the behavior for a specified hardkey (default = 0: no toggle).
<a href="#">SIM_HARDKEY_SetState()</a>	Set the state for a specified hardkey.

## SIM\_SetLCDPos()

### Description

Sets the position for the simulated LCD within the target device bitmap.

### Prototype

```
void SIM_SetLCDPos(int x, int y);
```

Parameter	Meaning
<a href="#">x</a>	X-position of the upper left corner for the simulated LCD (in pixels).
<a href="#">y</a>	Y-position of the upper left corner for the simulated LCD (in pixels).

### Additional information

The X- and Y-positions are relative to the target device bitmap, therefore position (0,0) refers to the upper left corner (origin) of the bitmap and not your actual LCD. Only the origin of the simulated screen needs to be specified; the resolution of your display should already be reflected in the configuration files in the `Config` directory.

## SIM\_SetTransColor()

### Description

Sets the color to be used for transparent areas of device or hardkey bitmaps.

### Prototype

```
I32 SIM_SetTransColor(I32 Color);
```

Parameter	Meaning
<a href="#">Color</a>	RGB value of the color.

### Additional information

The default setting for transparency is bright red (0xFF0000).

You would typically only need to change this setting if your bitmap contains the same shade of red.

# SIM\_HARDKEY\_GetNum()

### Description

Returns the number of available hardkeys.

### Prototype

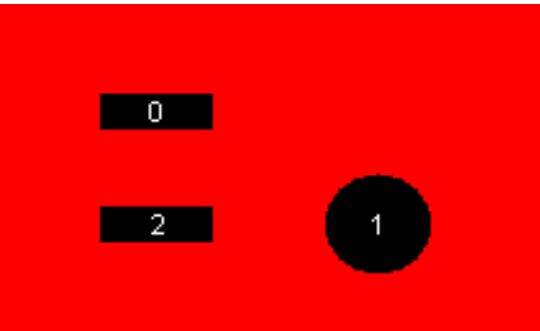
```
int SIM_HARDKEY_GetNum(void);
```

### Return value

The number of available hardkeys found in the bitmap.

### Additionalnnaal information

The numbering order for hardkeys is standard reading order (left to right, then top to bottom). The topmost pixel of a hardkey is therefore found first, regardless of its horizontal position. In the bitmap below, for example, the hardkeys are labeled as they would be referenced by the [KeyIndex](#) parameter in other functions:



It is recommended to call this function in order to verify that a bitmap is properly loaded.

# SIM\_HARDKEY\_GetState()

### Description

Returns the state of a specified hardkey.

### Prototype

```
int SIM_HARDKEY_GetState(unsigned int KeyIndex);
```

Parameter	Meaning
<a href="#">KeyIndex</a>	Index of hardkey (0 = index of first key).

### Return value

State of the specified hardkey:  
0: unpressed  
1: pressed

# SIM\_HARDKEY\_SetCallback()

### Description

Sets a callback routine to be executed when the state of a specified hardkey changes.

### Prototype

```
SIM_HARDKEY_CB* SIM_HARDKEY_SetCallback(unsigned int KeyIndex,
```

```
SIM_HARDKEY_CB* pfCallback);
```

Parameter	Meaning
<a href="#">KeyIndex</a>	Index of hardkey (0 = index of first key).
<a href="#">pfCallback</a>	Pointer to callback routine.

### Return value

Pointer to the previous callback routine.

### Additional information

The callback routine must have the following prototype:

### Prototype

```
typedef void SIM_HARDKEY_CB(int KeyIndex, int State);
```

Parameter	Meaning
<a href="#">KeyIndex</a>	Index of hardkey (0 = index of first key).
<a href="#">State</a>	State of the specified hardkey (see table below).

Permitted values for parameter <a href="#">State</a>	
0	Unpressed.
1	Pressed.

## SIM\_HARDKEY\_SetMode()

### Description

Sets the behavior for a specified hardkey.

### Prototype

```
int SIM_HARDKEY_SetMode(unsigned int KeyIndex, int Mode);
```

Parameter	Meaning
<a href="#">KeyIndex</a>	Index of hardkey (0 = index of first key).
<a href="#">Mode</a>	Behavior mode (see table below).

Permitted values for parameter <a href="#">Mode</a>	
0	Normal behavior (default).
1	Toggle behavior.

### Additional information

Normal (default) hardkey behavior means that a key is considered pressed only as long as the mouse button is held down on it. When the mouse is released or moved off of the hardkey, the key is considered unpressed.

With toggle behavior, each click of the mouse toggles the state of a hardkey to pressed or unpressed. That means if you click the mouse on a hardkey and it becomes pressed, it will remain pressed until you click the mouse on it again.

## SIM\_HARDKEY\_SetState()

### Description

Sets the state for a specified hardkey.

### Prototype

```
int SIM_HARDKEY_SetState(unsigned int KeyIndex, int State);
```

Parameter	Meaning
<a href="#">KeyIndex</a>	Index of hardkey (0 = index of first key).
<a href="#">State</a>	State of the specified hardkey (see table below).

Permitted values for parameter <a href="#">State</a>	
0	Unpressed.
1	Pressed.

### Additional information

This function is only usable when `SIM_HARDKEY_SetMode` is set to 1 (toggle mode).

# Chapter 4

## Displaying Text

---

It is very easy to display text with  $\mu$ C/GUI. Knowledge of only a few routines already allows you to write any text, in any available font, at any point on the display. We first provide a short introduction to displaying text, followed by more detailed explanations of the individual routines that are available.

# 4.1 Basic routines

In order to display text on the LCD, simply call the routine `GUI_DispString()` with the text you want to display as parameters. For example:

```
GUI_DispString("Hello world!");
```

The above code will display the text "Hello world" at the current text position. However, as you will see, there are routines to display text in a different font or in a certain position. In addition, it is possible to write not only strings but also decimal, hexadecimal and binary values to the display. Even though the graphic displays are usually byte-oriented, the text can be positioned at any pixel of the display, not only at byte positions.

## Control characters

Control characters are characters with a character code of less than 32. The control characters are defined as part of ASCII. `µC/GUI` ignores all control characters except for the following:

Char. Code	ASCII code	"C"	Meaning
10	LF	\n	Line feed. The current text position is changed to the beginning of the next line. Per default, this is: X = 0. Y + =font-distance in pixels (as delivered by <code>GUI_GetFontDistY()</code> ).
13	CR	\r	Carriage return. The current text position is changed to the beginning of the current line. Per default, this is: X = 0.

Usage of the control character `LF` can be very convenient in strings. A line feed can be made part of a string so that a string spanning multiple lines can be displayed with a single routine call.

## Positioning text at a selected position

This may be done by using the routine `GUI_GotoXY()` as shown in the following example:

```
GUI_GotoXY(10,10);// Set text position (in pixels)
GUI_DispString("Hello world!");// Show text
```

# 4.2 Text API

The table below lists the available text-related routines in alphabetical order within their respective categories. Detailed descriptions of the routines can be found in the sections that follow.

Routine	Explanation
Routines to display text	
<code>GUI_DispChar()</code>	Display single character at current position.
<code>GUI_DispCharAt()</code>	Display single character at specified position.
<code>GUI_DispChars()</code>	Display character a specified number of times.
<code>GUI_DispString()</code>	Display string at current position.
<code>GUI_DispStringAt()</code>	Display string at specified position.

Routine	Explanation
<a href="#">GUI_DispStringAtCEOL()</a>	Display string at specified position, then clear to end of line.
<a href="#">GUI_DispStringInRect()</a>	Display string in specified rectangle.
<a href="#">GUI_DispStringLen()</a>	Display string at current position with specified number of characters.
Selecting text drawing modes	
<a href="#">GUI_SetTextMode()</a>	Set text drawing mode.
Selecting text alignment	
<a href="#">GUI_GetTextAlign()</a>	Return current text alignment mode.
<a href="#">GUI_SetLBorder()</a>	Set left border after line feed.
<a href="#">GUI_SetTextAlign()</a>	Set text alignment mode.
Setting the current text position	
<a href="#">GUI_GotoX()</a>	Set current X-position.
<a href="#">GUI_GotoXY()</a>	Set current (X,Y) position.
<a href="#">GUI_GotoY()</a>	Set current Y-position.
Retrieving the current text position	
<a href="#">GUI_GetDispPosX()</a>	Return current X-position.
<a href="#">GUI_GetDispPosY()</a>	Return current Y-position.
Routines to clear a window or parts of it	
<a href="#">GUI_Clear()</a>	Clear active window (or entire display if background is the active window).
<a href="#">GUI_DispCEOL()</a>	Clear display from current text position to end of line.

## 4.3 Routines to display text

### GUI\_DispChar()

#### Description

Displays a single character at the current text position in the current window using the current font.

#### Prototype

```
void GUI_DispChar(U16 c);
```

Parameter	Meaning
<a href="#">c</a>	Character to display.

#### Additional information

This is the basic routine for displaying a single character. All other display routines ([GUI\\_DispCharAt\(\)](#), [GUI\\_DispString\(\)](#), etc.) call this routine to output the individual characters.

Which characters are available depends on the selected font. If the character is not available in the current font, nothing is displayed.

#### Example

Shows a capital A on the display:

```
GUI_DispChar('A');
```

#### Related topics

[GUI\\_DispChars\(\)](#), [GUI\\_DispCharAt\(\)](#)

## GUI\_DispCharAt()

### Description

Displays a single character at a specified position in the current window using the current font.

### Prototype

```
void GUI_DispCharAt(U16 c, I16P x, I16P y);
```

Parameter	Meaning
<a href="#">c</a>	Character to display.
<a href="#">x</a>	X-position to write to in pixels of the client window.
<a href="#">y</a>	Y-position to write to in pixels of the client window.

### Add information

Displays the character with its upper left corner at the specified (X,Y) position. Writes the character using the routine `GUI_DispChar()`. If the character is not available in the current font, nothing is displayed.

### Example

Shows a capital A on the display in the upper left corner:

```
GUI_DispCharAt('A',0,0);
```

### Related topics

`GUI_DispChar()`, `GUI_DispChars()`

## GUI\_DispChars()

### Description

Displays a character a specified number of times at the current text position in the current window using the current font.

### Prototype

```
void GUI_DispChars(U16 c, int Cnt);
```

Parameter	Meaning
<a href="#">c</a>	Character to display.
<a href="#">Cnt</a>	Number of repetitions (0 <= Cnt <= 32767).

### Additional information

Writes the character using the routine `GUI_DispChar()`. If the character is not available in the current font, nothing is displayed.

### Example

Shows the line "\*\*\*\*\*" on the display:

```
GUI_DispChars('*', 30);
```

### Related topics

`GUI_DispChar()`, `GUI_DispCharAt()`

## GUI\_DispString()

### Description

Displays the string passed as parameter at the current text position in the current window using the current font.

### Prototype

```
void GUI_DispString(const char GUI_FAR *s);
```

Parameter	Meaning
<a href="#">s</a>	String to display.

### Additional information

The string can contain the control character `\n`. This control character moves the current text position to the beginning of the next line.

### Example

Shows "Hello world" on the display and "Next line" on the next line:

```
GUI_DispString("Hello world");// Disp text
GUI_DispString("\nNext line");// Disp text
```

### Related topics

[GUI\\_DispStringAt\(\)](#), [GUI\\_DispStringAtCEOL\(\)](#), [GUI\\_DispStringLen\(\)](#)

## GUI\_DispStringAt()

### Description

Displays the string passed as parameter at a specified position in the current window using the current font.

### Prototype

```
void GUI_DispStringAt(const char GUI_FAR *s, int x, int y);
```

Parameter	Meaning
<a href="#">s</a>	String to display.
<a href="#">x</a>	X-position to write to in pixels of the client window.
<a href="#">y</a>	Y-position to write to in pixels of the client window.

### Example

Shows "Position 50,20" at position 50,20 on the display:

```
GUI_DispStringAt("Position 50,20", 50, 20);// Disp text
```

### Related topics

[GUI\\_DispString\(\)](#), [GUI\\_DispStringAtCEOL\(\)](#), [GUI\\_DispStringLen\(\)](#)

## GUI\_DispStringAtCEOL()

### Description

This routine uses the exact same parameters as [GUI\\_DispStringAt\(\)](#). It does the same thing: displays a given string at a specified position. However, after doing so, it clears the remaining part of the line to the end by calling the routine [GUI\\_DispCEOL\(\)](#). This routine can be handy if one string is to overwrite another, and the overwriting string is or may be shorter than the previous one.

# GUI\_DispStringInRect()

## Description

Displays the string passed as parameter at a specified position within a specified rectangle, in the current window using the current font.

## Prototype

```
void GUI_DispStringInRect(const char GUI_FAR *s, const GUI_RECT *pRect,
int Align);
```

Parameter	Meaning
s	String to display.
pRect	Rectangle to write to in pixels of the client window.
Align	Alignment flags; "OR" combinable. A flag for horizontal and a flag for vertical alignment should be combined. Available flags are: GUI_TA_TOP, GUI_TA_BOTTOM, GUI_TA_VCENTER for vertical alignment. GUI_TA_LEFT, GUI_TA_RIGHT, GUI_TA_HCENTER for horizontal alignment.

## Example

Shows the word "Text" centered horizontally and vertically in the current window:

```
GUI_RECT rClient;
GUI_GetClientRect(&rClient);
GUI_DispStringInRect("Text", &rClient, GUI_TA_HCENTER | GUI_TA_VCENTER);
```

## Additionaln information

If the specified rectangle is too small, the text will be clipped.

## Related topics

```
GUI_DispString(), GUI_DispStringAtCEOL(), GUI_DispStringLength()
```

# GUI\_DispStringLength()

## Description

Displays the string passed as parameter with a specified number of characters at the current text position, in the current window using the current font.

## Prototype

```
void GUI_DispStringLength(const char GUI_FAR *s, int Len);
```

Parameter	Meaning
s	String to display. Should be a \0 terminated array of 8-bit character. Passing NULL as parameter is permitted.
Len	Number of characters to display.

## Additionaln information

If the string has less characters than specified (is shorter), it is padded with spaces. If the string has more characters than specified (is longer), then only the given number of characters is actually displayed. This function is especially useful if text messages can be displayed in different languages (and will naturally differ in length), but only a certain number of characters can be displayed.

## Related topics

```
GUI_DispString(), GUI_DispStringAt(), GUI_DispStringAtCEOL()
```

## 4.4 Selecting text drawing modes

Normally, text is written into the selected window at the current text position using the selected font in normal text. Normal text means that the text overwrites whatever is already displayed where the bits set in the character mask are set on the display. In this mode, active bits are written using the foreground color, while inactive bits are written with the background color. However, in some situations it may be desirable to change this default behavior.  $\mu$ C/GUI offers four flags for this purpose (one default plus three modifiers), which may be combined:

### Normal text

Text can be displayed normally by specifying `GUI_TEXTMODE_NORMAL` or 0.

### Reverse text

Text can be displayed in reverse by specifying `GUI_TEXTMODE_REVERSE`. What is usually displayed as white on black will be displayed as black on white.

### Transparent text

Transparent text means that the text is written on top of whatever is already visible on the display. The difference is that whatever was previously on the screen can still be seen, whereas with normal text the background is erased.

Text can be displayed transparently by specifying `GUI_TEXTMODE_TRANS`.

### XOR text

What usually is drawn white (the actual character) is inverted. The effect is identical to that of the default mode (normal text) if the background is black. If the background is white, the output is identical to reverse text.

If you use colors, an inverted pixel is calculated as follows:

New pixel color = number of colors - actual pixel color - 1.

### Transparent reversed text

As with transparent text, it does not overwrite the background, and as with reverse text, the text is displayed in reverse.

Text can be displayed in reverse transparently by specifying `GUI_TEXTMODE_TRANS | GUI_TEXTMODE_REVERSE`.

### Example

Displays normal, reverse, transparent, XOR, and transparent reversed text:

```
GUI_SetFont(&GUI_Font8x16);
GUI_SetFont(&GUI_Font8x16);
GUI_SetBkColor(GUI_BLUE);
GUI_Clear();
GUI_SetPenSize(10);
GUI_SetColor(GUI_RED);
GUI_DrawLine(80, 10, 240, 90);
GUI_DrawLine(80, 90, 240, 10);
GUI_SetBkColor(GUI_BLACK);
GUI_SetColor(GUI_WHITE);
GUI_SetTextMode(GUI_TM_NORMAL);
GUI_DispStringHCenterAt("GUI_TM_NORMAL", 160, 10);
GUI_SetTextMode(GUI_TM_REV);
GUI_DispStringHCenterAt("GUI_TM_REV", 160, 26);
GUI_SetTextMode(GUI_TM_TRANS);
GUI_DispStringHCenterAt("GUI_TM_TRANS", 160, 42);
GUI_SetTextMode(GUI_TM_XOR);
GUI_DispStringHCenterAt("GUI_TM_XOR", 160, 58);
GUI_SetTextMode(GUI_TM_TRANS | GUI_TM_REV);
GUI_DispStringHCenterAt("GUI_TM_TRANS | GUI_TM_REV", 160, 74);
```

Screen shot of above example



GUI\_SetTextMode()

Description

Sets the text mode to the parameter specified.

Prototype

```
int GUI_SetTextMode(int TextMode);
```

Parameter	Meaning
TextMode	Text mode to set. May be any combination of the TEXTMODE flags.

Permitted values for parameter TextMode (OR-combinable)	
GUI_TEXTMODE_NORMAL	Sets normal text. This is the default setting; the value is identical to 0.
GUI_TEXTMODE_REVERSE	Sets reverse text.
GUI_TEXTMODE_TRANSPARENT	Sets transparent text.
GUI_TEXTMODE_XOR	Text will be inverted on the display.

Return value

The selected text mode.

Example

Shows "The value is" at position 0,0 on the display, shows a value in reverse text, then sets the text mode back to normal:

```
int i = 20;
GUI_DispStringAt("The value is", 0, 0);
GUI_SetTextMode(GUI_TEXTMODE_REVERSE);
GUI_DispDec(20, 3);
GUI_SetTextMode(GUI_TEXTMODE_NORMAL);
```

4.5 Selecting text alignment

GUI\_GetTextAlign()

Description

Returns the current text alignment mode.

Prototype

```
int GUI_GetTextAlign(void);
```

## GUI\_SetLBorder()

### Description

Sets the left border for line feeds in the current window.

### Prototype

```
void GUI_SetLBorder(int x)
```

Parameter	Meaning
<code>x</code>	New left border (in pixels, 0 is left border).

## GUI\_SetTextAlign()

### Description

Sets the text alignment mode for string output in the current window.

### Prototype

```
int GUI_SetTextAlign(int TextAlign);
```

Parameter	Meaning
<code>TextAlign</code>	Text alignment mode to set. May be a combination of a horizontal and a vertical alignment flag.

Permitted values for parameter <code>TextAlign</code> (horizontal and vertical flags are OR-combinable)	
Horizontal alignment	
<code>GUI_TA_LEFT</code>	Align X-position left (default).
<code>GUI_TA_HCENTER</code>	Center X-position.
<code>GUI_TA_RIGHT</code>	Align X-position right (default).
Vertical alignment	
<code>GUI_TA_TOP</code>	Align Y-position with top of characters (default).
<code>GUI_TA_VCENTER</code>	Center Y-position.
<code>GUI_TA_BOTTOM</code>	Align Y-position with bottom pixel line of font.

### Return value

The selected text alignment mode.

### Additionaln information

`GUI_SetTextAllign()` does not affect the character output routines beginning with `GUI_DispChar()`.

### Example

Displays the value 1234 with the center of the text at x=100, y=100:

```
GUI_SetTextAlign(GUI_TA_HCENTER | GUI_TA_VCENTER);
GUI_DispDecAt(1234,100,100,4);
```

# 4.6 Setting the current text position

Every task has a current text position. This is the position relative to the origin of the window (usually (0,0)) where the next character will be written if a text output routine is called. Initially, this position is (0,0), which is the upper left corner of the current window. There are 3 functions which can be used to set the current text position.

## GUI\_GotoXY(), GUI\_GotoX(), GUI\_GotoY()

### Description

Set the current text write position.

### Prototypes

```
char GUI_GotoXY(int x, int y);
char GUI_GotoX(int x);
char GUI_GotoY(int y);
```

Parameter	Meaning
x	New X-position (in pixels, 0 is left border).
y	New Y-position (in pixels, 0 is top border).

### Return value

Usually 0.  
If a value != 0 is returned, then the current text position is outside of the window (to the right or below), so a following write operation can be omitted.

### Additionaln information

GUI\_GotoXY() sets both the X- and Y-components of the current text position.  
GUI\_GotoX() sets the X-component of the current text position; the Y-component remains unchanged.  
GUI\_GotoY() sets the Y-component of the current text position; the X-component remains unchanged.

### Example

Shows "(20,20)" at position 20,20 on the display:

```
GUI_GotoXY(20,20)
GUI_DispString("The value is");
```

# 4.7 Retrieving the current text position

## GUI\_GetDispPosX()

### Description

Returns the current X-position.

### Prototype

```
int GUI_GetDispPosX(void);
```

## GUI\_GetDispPosY()

### Description

Returns the current Y-position.

### Prototype

```
int GUI_GetDispPosY(void);
```

## 4.8 Routines to clear a window or parts of it

### GUI\_Clear()

#### Description

Clears the current window.

#### Prototype

```
void GUI_Clear(void);
```

#### Additional information

If no window has been defined, the current window is the entire display. In this case, the entire display is cleared.

#### Example

Shows "Hello world" on the display, waits 1 second and then clears the display:

```
GUI_DispStringAt("Hello world", 0, 0); // Disp text
GUI_Delay(1000); // Wait 1 second (not part of µC/GUI)
GUI_Clear(); // Clear screen
```

### GUI\_DispCEOL()

#### Description

Clears the current window (or the display) from the current text position to the end of the line using the height of the current font.

#### Prototype

```
void GUI_DispCEOL(void);
```

#### Example

Shows "Hello world" on the display, waits 1 second and then displays "Hi" in the same place, replacing the old string:

```
GUI_DispStringAt("Hello world", 0, 0); // Disp text
Delay (1000);
GUI_DispStringAt("Hi", 0, 0);
GUI_DispCEOL();
```



# Chapter 5

## Displaying Values

---

The preceding chapter explained how to show strings on the display. Of course you may use strings and the functions of the standard "C" library to display values. However, this can sometimes be a difficult task. It is usually much easier (and much more efficient) to call a routine that displays the value in the form that you want.  $\mu$ C/GUI supports different decimal, hexadecimal and binary outputs. The individual routines are explained in this chapter.

All functions work without the usage of a floating-point library and are optimized for both speed and size. Of course `sprintf` may also be used on any system. Using the routines in this chapter can sometimes simplify things and save both ROM space and execution time.

## 5.1 Value API

The table below lists the available value-related routines in alphabetical order within their respective categories. Detailed descriptions of the routines can be found in the sections that follow.

Routine	Explanation
Displaying decimal values	
<code>GUI_DispDec()</code>	Display value in decimal form at current position with specified number of characters.
<code>GUI_DispDecAt()</code>	Display value in decimal form at specified position with specified number of characters.
<code>GUI_DispDecMin()</code>	Display value in decimal form at current position with minimum number of characters.
<code>GUI_DispDecShift()</code>	Display long value in decimal form with decimal point at current position with specified number of characters.
<code>GUI_DispDecSpace()</code>	Display value in decimal form at current position with specified number of characters, replace leading zeros with spaces.
<code>GUI_DispSDec()</code>	Display value in decimal form at current position with specified number of characters and sign.
<code>GUI_DispSDecShift()</code>	Display long value in decimal form with decimal point at current position with specified number of characters and sign.
Displaying floating-point values	
<code>GUI_DispFloat()</code>	Display floating-point value with specified number of characters.
<code>GUI_DispFloatFix()</code>	Display floating-point value with fixed no. of digits to the right of decimal point.
<code>GUI_DispFloatMin()</code>	Display floating-point value with minimum number of characters.
<code>GUI_DispSFloatFix()</code>	Display floating-point value with fixed no. of digits to the right of decimal point and sign.
<code>GUI_DispSFloatMin()</code>	Display floating-point value with minimum number of characters and sign.
Displaying binary values	
<code>GUI_DispBin()</code>	Display value in binary form at current position.
<code>GUI_DispBinAt()</code>	Display value in binary form at specified position.
Displaying hexadecimal values	
<code>GUI_DispHex()</code>	Display value in hexadecimal form at current position.
<code>GUI_DispHexAt()</code>	Display value in hexadecimal form at specified position.

## 5.2 Displaying decimal values

### GUI\_DispDec()

#### Description

Displays a value in decimal form with a specified number of characters at the current text position, in the current window using the current font.

## Prototype

```
void GUI_DisDec(I32 v, U8 Len);
```

Parameter	Meaning
<a href="#">v</a>	Value to display. Minimum -2147483648 (= $-2^{31}$ ). Maximum 2147483647 (= $2^{31} - 1$ ).
<a href="#">Len</a>	No. of digits to display (max. 9).

## Additional information

Leading zeros are not suppressed (are shown as 0).  
If the value is negative, a minus sign is shown.

## Example

```
// Display time as minutes and seconds
GUI_DisString("Min:");
GUI_DisDec(Min,2);
GUI_DisString(" Sec:");
GUI_DisDec(Sec,2);
```

## Related topics

[GUI\\_DisDec\(\)](#), [GUI\\_DisDecAt\(\)](#), [GUI\\_DisDecMin\(\)](#), [GUI\\_DisDecSpace\(\)](#)

## GUI\_DisDecAt()

### Description

Displays a value in decimal form with a specified number of characters at a specified position, in the current window using the current font.

### Prototype

```
void GUI_DisDecAt(I32 v, I16P x, I16P y, U8 Len);
```

Parameter	Meaning
<a href="#">v</a>	Value to display. Minimum -2147483648 (= $-2^{31}$ ). Maximum 2147483647 (= $2^{31} - 1$ ).
<a href="#">x</a>	X-position to write to in pixels of the client window.
<a href="#">y</a>	Y-position to write to in pixels of the client window.
<a href="#">Len</a>	No. of digits to display (max. 9).

## Additional information

Leading zeros are not suppressed.  
If the value is negative, a minus sign is shown.

## Example

```
// Update seconds in upper right corner
GUI_DisDecAT(Sec, 200, 0, 2);
```

## Related topics

[GUI\\_DisDec\(\)](#), [GUI\\_DisDecMin\(\)](#), [GUI\\_DisDecSpace\(\)](#)

## GUI\_DispDecMin()

### Description

Displays a value in decimal form at the current text position in the current window using the current font. The length need not be specified; the minimum length will automatically be used.

### Prototype

```
void GUI_DispDecMin(I32 v);
```

Parameter	Meaning
<code>v</code>	Value to display. Minimum: -2147483648 ( $= -2^{31}$ ); maximum 2147483647 ( $= 2^{31} - 1$ ). Maximum no. of digits displayed is 9.

### Additional information

If values have to be aligned but differ in the number of digits, this function is not a good choice. Try one of the functions that specify the number of digits.

### Example

```
// Show result
GUI_DispString("The result is :");
GUI_DispDecMin(Result);
```

### Related topics

GUI\_DispDec(), GUI\_DispDecAt(), GUI\_DispSDec(), GUI\_DispDecSpace()

## GUI\_DispDecShift()

### Description

Displays a `long` value in decimal form with a specified number of characters and with decimal point at the current text position, in the current window using the current font.

### Prototype

```
void GUI_DispDecShift(I32 v, U8 Len, U8 Shift);
```

Parameter	Meaning
<code>v</code>	Value to display. Minimum: -2147483648 ( $= -2^{31}$ ); maximum: 2147483647 ( $= 2^{31} - 1$ ).
<code>Len</code>	No. of digits to display (max. 9).
<code>Shift</code>	No. of digits to show to right of decimal point.

### Additional information

Watch the maximum number of 9 characters (including sign and decimal point).

## GUI\_DispDecSpace()

### Description

Displays a value in decimal form at the current text position in the current window using the current font. Leading zeros are suppressed (replaced by spaces).

## Prototype

```
void DispDecSpace(I32 v, U8 MaxDigits);
```

Parameter	Meaning
<a href="#">v</a>	Value to display. Minimum: -2147483648 ( $= -2^{31}$ ); maximum: 2147483647 ( $= 2^{31} - 1$ ).
<a href="#">MaxDigits</a>	No. of digits to display, including leading spaces. Maximum no. of digits displayed is 9 (excluding leading spaces).

## Additional information

If values have to be aligned but differ in the number of digits, this function is a good choice.

## Example

```
// Show result
GUI_DisString("The result is :");
GUI_DisDecSpace(Result, 200);
```

## Related topics

[GUI\\_DisDec\(\)](#), [GUI\\_DisDecAt\(\)](#), [GUI\\_DisSDec\(\)](#), [GUI\\_DisDecMin\(\)](#)

## GUI\_DisSDec()

### Description

Displays a value in decimal form (with sign) with a specified number of characters at the current text position, in the current window using the current font.

### Prototype

```
void GUI_DisSDec(I32 v, U8 Len);
```

Parameter	Meaning
<a href="#">v</a>	Value to display. Minimum: -2147483648 ( $= -2^{31}$ ); maximum: 2147483647 ( $= 2^{31} - 1$ ).
<a href="#">Len</a>	No. of digits to display (max. 9).

## Additional information

Leading zeros are not suppressed.

This function is similar to [GUI\\_DisDec](#), but a sign is always shown in front of the value, even if the value is positive.

## Related topics

[GUI\\_DisDec\(\)](#), [GUI\\_DisDecAt\(\)](#), [GUI\\_DisDecMin\(\)](#), [GUI\\_DisDecSpace\(\)](#)

## GUI\_DisSDecShift()

### Description

Displays a `long` value in decimal form (with sign) with a specified number of characters and with decimal point at the current text position, in the current window using the current font.

## Prototype

```
void GUI_DispNetDecShift(I32 v, U8 Len, U8 Shift);
```

Parameter	Meaning
<a href="#">v</a>	Value to display. Minimum: -2147483648 (= $-2^{31}$ ); maximum: 2147483647 (= $2^{31} - 1$ ).
<a href="#">Len</a>	No. of digits to display (max. 9).
<a href="#">Shift</a>	No. of digits to show to right of decimal point.

## Additional information

A sign is always shown in front of the value.

Watch the maximum number of 9 characters (including sign and decimal point).

## Example

```
void DemoDec(void) {
    long l = 12345;
    GUI_Clear();
    GUI_SetFont(&GUI_Font8x8);
    GUI_DispNetStringAt("GUI_DispNetDecShift:\n",0,0);
    GUI_DispNetDecShift(l, 7, 3);
    GUI_SetFont(&GUI_Font6x8);
    GUI_DispNetStringAt("Press any key",0,GUI_VYSIZE-8);
    WaitKey();
}
```

## Screen shot of above example



# 5.3 Displaying floating-point values

## GUI\_DispNetFloat()

### Description

Displays a floating-point value with a specified number of characters at the current text position in the current window using the current font.

### Prototype

```
void GUI_DispNetFloat(float v, char Len);
```

Parameter	Meaning
<a href="#">v</a>	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
<a href="#">Len</a>	No. of digits to display (max. 9).

## Additional information

Leading zeros are suppressed. The decimal point counts as one character.

If the value is negative, a minus sign is shown.

### Example

```
/* Shows all features for displaying floating point values */
void DemoFloat(void) {
    float f = 123.45678;
    GUI_Clear();
    GUI_SetFont(&GUI_Font8x8);
    GUI_DispStringAt("GUI_DisFloat:\n",0,0);
    GUI_DisFloat (f,9);
    GUI_GotoX(100);
    GUI_DisFloat (-f,9);
    GUI_DisStringAt("GUI_DisFloatFix:\n",0,20);
    GUI_DisFloatFix (f,9,2);
    GUI_GotoX(100);
    GUI_DisFloatFix (-f,9,2);
    GUI_DisStringAt("GUI_DisSFloatFix:\n",0,40);
    GUI_DisSFloatFix (f,9,2);
    GUI_GotoX(100);
    GUI_DisSFloatFix (-f,9,2);
    GUI_DisStringAt("GUI_DisFloatMin:\n",0,60);
    GUI_DisFloatMin (f,3);
    GUI_GotoX(100);
    GUI_DisFloatMin (-f,3);
    GUI_DisStringAt("GUI_DisSFloatMin:\n",0,80);
    GUI_DisSFloatMin (f,3);
    GUI_GotoX(100);
    GUI_DisSFloatMin (-f,3);
    GUI_SetFont(&GUI_Font6x8);
    GUI_DisStringAt("Press any key",0,GUI_VYSIZE-8);
    WaitKey();
}
```

### Screen shot of above example



```
GUI_DisFloat:
123.45678      -123.4568
GUI_DisFloatFix:
000123.46      -00123.46
GUI_DisSFloatFix:
+00123.46      -00123.46
GUI_DisFloatMin:
123.457        -123.457
GUI_DisSFloatMin:
+123.457        -123.457

Press any key
```

## GUI\_DisFloatFix()

### Description

Displays a floating-point value with specified number of total characters and a specified number of characters to the right of the decimal point, at the current text position in the current window using the current font.

## Prototype

```
void GUI_DisFloatFix (float v, char Len, char Decs);
```

Parameter	Meaning
<a href="#">v</a>	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
<a href="#">Len</a>	No. of digits to display (max. 9).
<a href="#">Decs</a>	No. of digits to show to right of decimal point.

## Additional information

Leading zeros are not suppressed.

If the value is negative, a minus sign is shown.

## GUI\_DisFloatMin()

### Description

Displays a floating-point value with a minimum number of decimals to the right of the decimal point, at the current text position in the current window using the current font.

### Prototype

```
void GUI_DisFloatMin(float f, char Fract);
```

Parameter	Meaning
<a href="#">v</a>	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
<a href="#">Fract</a>	Minimum no. of characters to display.

## Additional information

Leading zeros are suppressed.

If the value is negative, a minus sign is shown.

The length need not be specified; the minimum length will automatically be used. If values have to be aligned but differ in the number of digits, this function is not a good choice. Try one of the functions that specify the number of digits.

## GUI\_DisSFloatFix()

### Description

Displays a floating-point value (with sign) with a specified number of total characters and a specified number of characters to the right of the decimal point, in the current window using the current font.

### Prototype

```
void GUI_DisSFloatFix(float v, char Len, char Decs);
```

Parameter	Meaning
<a href="#">v</a>	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
<a href="#">Len</a>	No. of digits to display (max. 9).
<a href="#">Decs</a>	No. of digits to show to right of decimal point.

### Additional information

Leading zeros are not suppressed.  
A sign is always shown in front of the value.

## GUI\_DispSFloatMin()

### Description

Displays a floating-point value (with sign) with a minimum number of decimals to the right of the decimal point, at the current text position in the current window using the current font.

### Prototype

```
void GUI_DispSFloatMin(float f, char Fract);
```

Parameter	Meaning
<code>v</code>	Value to display. Minimum 1.2 E-38; maximum 3.4 E38.
<code>Fract</code>	Minimum no. of digits to display.

### Additional information

Leading zeros are suppressed.  
A sign is always shown in front of the value.  
The length need not be specified; the minimum length will automatically be used. If values have to be aligned but differ in the number of digits, this function is not a good choice. Try one of the functions that specify the number of digits.

## 5.4 Displaying binary values

### GUI\_DispBin()

#### Description

Displays a value in binary form at the current text position in the current window using the current font.

#### Prototype

```
void GUI_DispBin(U32 v, U8 Len);
```

Parameter	Meaning
<code>v</code>	Value to display, 32-bit.
<code>Len</code>	No. of digits to display (including leading zeros).

### Additional information

As with decimal and hexadecimal values, the least significant bit is rightmost.

### Example

```
//
// Show binary value 7, result: 000111
//
U32 Input = 0x7;
GUI_DispBin(Input, 6);
```

## Related topics

GUI\_DispBinAt()

## GUI\_DispBinAt()

### Description

Displays a value in binary form at a specified position in the current window using the current font.

### Prototype

```
void DispBinAt(U32 v, I16P y, I16P x, U8 Len);
```

Parameter	Meaning
<a href="#">v</a>	Value to display, 16-bit.
<a href="#">x</a>	X-position to write to in pixels of the client window.
<a href="#">y</a>	Y-position to write to in pixels of the client window.
<a href="#">Len</a>	No. of digits to display (including leading zeroes).

### Additional information

As with decimal and hexadecimal values, the least significant bit is rightmost.

### Example

```
//
// Show binary input status
//
GUI_DispBinAt(Input, 0,0, 8);
```

### Related topics

GUI\_DispBin(), GUI\_DispHex()

## 5.5 Displaying hexadecimal values

### GUI\_DispHex()

#### Description

Displays a value in hexadecimal form at the current text position in the current window using the current font.

#### Prototype

```
void GUI_DispHex(U32 v, U8 Len);
```

Parameter	Meaning
<a href="#">v</a>	Value to display, 16-bit.
<a href="#">Len</a>	No. of digits to display.

#### Additional information

As with decimal and binary values, the least significant bit is rightmost.

## Example

```
/* Show value of AD-converter */
GUI_DispHex(Input, 4);
```

## Related topics

GUI\_DispDec(), GUI\_DispBin(), GUI\_DispHexAt()

## GUI\_DispHexAt()

### Description

Displays a value in hexadecimal form at a specified position in the current window using the current font.

### Prototype

```
void GUI_DispHexAt(U32 v, I16P x, I16P y, U8 Len);
```

Parameter	Meaning
<a href="#">v</a>	Value to display, 16-bit.
<a href="#">x</a>	X-position to write to in pixels of the client window.
<a href="#">y</a>	Y-position to write to in pixels of the client window.
<a href="#">Len</a>	No. of digits to display.

### Additional information

As with decimal and binary values, the least significant bit is rightmost.

### Example

```
//
// Show value of AD-converter at specified position
//
GUI_DispHexAt(Input, 0, 0, 4);
```

## Related topics

GUI\_DispDec(), GUI\_DispBin(), GUI\_DispHex()

## 5.6 Version of $\mu$ C/GUI

### GUI\_GetVersionString()

#### Description

Returns a string containing the current version of  $\mu$ C/GUI

#### Prototype

```
const char * GUI_GetVersionString(void);
```

#### Example

```
//
// Displays the current version at the current cursor position
//
GUI_DispString(GUI_GetVersionString());
```



# Chapter 6

## 2-D Graphic Library

---

$\mu$ C/GUI contains a complete 2-D graphic library which should be sufficient for most applications. The routines supplied with  $\mu$ C/GUI can be used with or without clipping (refer to Chapter 12: "Window Manager") and are based on fast and efficient algorithms. Currently, only the `DrawArc` set of functions requires floating-point calculations.

## 6.1 API reference: graphics

The table below lists the available graphic-related routines in alphabetical order within their respective categories. Detailed descriptions can be found in the sections that follow.

Routine	Explanation
Drawing modes	
<a href="#">GUI_SetDrawMode</a>	Set the drawing mode.
Basic drawing routines	
<a href="#">GUI_ClearRect</a>	Fill a rectangular area with the background color.
<a href="#">GUI_DrawPixel</a>	Draw a single pixel.
<a href="#">GUI_DrawPoint</a>	Draw a point.
<a href="#">GUI_FillRect</a>	Draw a filled rectangle.
<a href="#">GUI_InvertRect</a>	Invert a rectangular area.
Drawing bitmaps	
<a href="#">GUI_DrawBitmap</a>	Draw a bitmap.
<a href="#">GUI_DrawBitmapExp</a>	Draw a bitmap.
<a href="#">GUI_DrawBitmapMag</a>	Draw a magnified bitmap.
Drawing lines	
<a href="#">GUI_DrawHLine</a>	Draw a horizontal line.
<a href="#">GUI_DrawLine</a>	Draw a line.
<a href="#">GUI_DrawLineRel</a>	Draw a line from the current position to endpoint by X- and Y-distance.
<a href="#">GUI_DrawLineTo</a>	Draws a line from the current position to endpoint by X- and Y-position.
<a href="#">GUI_DrawPolyLine</a>	Draw a polyline.
<a href="#">GUI_DrawVLine</a>	Draw a vertical line.
Drawing polygons	
<a href="#">GUI_DrawPolygon</a>	Draw a polygon.
<a href="#">GUI_EnlargePolygon</a>	Enlarge a polygon.
<a href="#">GUI_FillPolygon</a>	Draw a filled polygon.
<a href="#">GUI_MagnifyPolygon</a>	Magnify a polygon.
<a href="#">GUI_RotatePolygon</a>	Rotate a polygon by a specified angle.
Drawing circles	
<a href="#">GUI_DrawCircle</a>	Draw a circle.
<a href="#">GUI_FillCircle</a>	Draw a filled circle.
Drawing ellipses	
<a href="#">GUI_DrawEllipse</a>	Draw an ellipse.
<a href="#">GUI_FillEllipse</a>	Draw a filled ellipse.
Drawing arcs	
<a href="#">GUI_DrawArc</a>	Draw an arc.

## 6.2 Drawing modes

μC/GUI can draw in NORMAL mode or in XOR mode. The default is NORMAL mode, in which the content of the display is overdrawn by the graphic. In XOR mode, the content of the display is inverted when it is overdrawn.

### Restrictions associated with GUI\_DRAWMODE\_XOR

- XOR mode is only useful when using two displayed colors inside the active window or screen.
- Some drawing functions of μC/GUI do not work precisely with this drawing mode. Generally, this mode works only with a pen size of one pixel. That means before using functions like `GUI_DrawLine`, `GUI_DrawCircle`, `GUI_DrawRect` and so on, you must make sure that the pen size is set to 1 when you are working in XOR mode.
- When drawing bitmaps with a color depth greater than 1 bit per pixel (bpp) this drawing mode has no effect.
- When using drawing functions such as `GUI_DrawPolyLine` or multiple calls of `GUI_DrawLineTo`, the fulcrums are inverted twice. The result is that these pixels remain in the background color.

## GUI\_SetDrawMode

### Description

Selects the specified drawing mode.

### Prototype

```
GUI_DRAWMODE GUI_SetDrawMode(GUI_DRAWMODE mode);
```

Parameter	Meaning
<code>mode</code>	Drawing mode to set. May be a value returned by any routine which sets the drawing mode or one of the constants below.

Permitted values for parameter <code>mode</code>	
<code>GUI_DRAWMODE_NORMAL</code>	Default: Draws points, lines, areas, bitmaps.
<code>GUI_DRAWMODE_XOR</code>	Inverts points, lines, areas when overwriting the color of another object on the display.

### Return value

The selected drawing mode.

### Additional Information

In addition to setting the drawing mode, this routine may also be used to restore a drawing mode that has previously been changed.

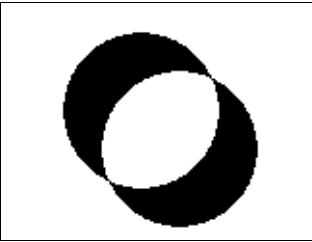
If using colors, an inverted pixel is calculated as follows:

New pixel color = number of colors - actual pixel color - 1.

### Example

```
//
// Showing two circles, the second one XOR-combined with the first:
//
GUI_Clear();
GUI_SetDrawMode(GUI_DRAWMODE_NORMAL);
GUI_FillCircle(120, 64, 40);
GUI_SetDrawMode(GUI_DRAWMODE_XOR);
GUI_FillCircle(140, 84, 40);
```

Screen shot for preceeding example



### 6.3 Basic drawing routines

The basic drawing routines allow drawing of individual points, horizontal and vertical lines and shapes at any position on the display. Any available drawing mode can be used. Since these routines are called frequently in most applications, they are optimized for speed as much as possible. For example, the horizontal and vertical line functions do not require the use of single-dot routines.

#### GUI\_ClearRect

**Description**

Clears a rectangular area at a specified position in the current window by filling it with the background color.

**Prototype**

```
void GUI_ClearRect(int x0, int y0, int x1, int y1);
```

Parameter	Meaning
x0	Upper left X-position.
y0	Upper left Y-position.
x1	Lower right X-position.
y1	Lower right Y-position.

**Related topics**

```
GUI_InvertRect, GUI_FillRect
```

#### GUI\_DrawPixel

**Description**

Draws a pixel at a specified position in the current window.

## Prototype

```
void GUI_DrawPixel(int x, int y);
```

Parameter	Meaning
<a href="#">x</a>	X-position of pixel.
<a href="#">y</a>	Y-position of pixel.

## Related topics

[GUI\\_DrawPoint](#)

## GUI\_DrawPoint

### Description

Draws a point with the current pen size at a specified position in the current window.

### Prototype

```
void GUI_DrawPoint(int x, int y);
```

Parameter	Meaning
<a href="#">x</a>	X-position of point.
<a href="#">y</a>	Y-position of point.

## Related topics

[GUI\\_DrawPixel](#)

## GUI\_FillRect

### Description

Draws a filled rectangular area at a specified position in the current window.

### Prototype

```
void GUI_FillRect(int x0, int y0, int x1, int y1);
```

Parameter	Meaning
<a href="#">x0</a>	Upper left X-position.
<a href="#">y0</a>	Upper left Y-position.
<a href="#">x1</a>	Lower right X-position.
<a href="#">y1</a>	Lower right Y-position.

### Additional Information

Uses the current drawing mode, which normally means all pixels inside the rectangle are set.

## Related topics

[GUI\\_InvertRect](#), [GUI\\_ClearRect](#)

## GUI\_InvertRect

### Description

Draws an inverted rectangular area at a specified position in the current window.

### Prototype

```
void GUI_InvertRect(int x0, int y0, int x1, int y1);
```

Parameter	Meaning
<a href="#">x0</a>	Upper left X-position.
<a href="#">y0</a>	Upper left Y-position.
<a href="#">x1</a>	Lower right X-position.
<a href="#">y1</a>	Lower right Y-position.

### Related topics

[GUI\\_FillRect](#), [GUI\\_ClearRect](#)

## 6.4 Drawing bitmaps

### GUI\_DrawBitmap

#### Description

Draws a bitmap image at a specified position in the current window.

#### Prototype

```
void GUI_DrawBitmap(const GUI_BITMAP*pBM, int x, int y);
```

Parameter	Meaning
<a href="#">pBM</a>	Pointer to the BITMAP to display.
<a href="#">x</a>	X-position of the upper left corner of the bitmap in the display.
<a href="#">y</a>	Y-position of the upper left corner of the bitmap in the display.

#### Additional Information

The bitmap data must be defined pixel by pixel. Every pixel is equivalent to one bit. The most significant bit (MSB) defines the first pixel; the picture data is interpreted as bitstream starting with the MSB of the first byte. A new line always starts at an even byte address, as the nth line of the bitmap starts at offset  $n \times \text{BytesPerLine}$ . The bitmap can be shown at any point in the client area. Usually, the bitmap converter is used to generate bitmaps.

#### Example

```
extern const GUI_BITMAP bmMicriumLogo; /* declare external Bitmap */

void main() {
    GUI_Init();
    GUI_DrawBitmap(&bmMicriumLogo, 45, 20);
}
```

## Screen shot for preceeding example

# Micrium

## GUI\_DrawBitmapExp

### Description

Same function as `GUI_DrawBitmap`, but with extended parameter settings.

### Prototype

```
void GUI_DrawBitmapExp(int x0, int y0,
                      int XSize, int YSize,
                      int XMul, int YMul,
                      int BitsPerPixel,
                      int BytesPerLine,
                      const U8* pData,
                      const GUI_LOGPALETTE* pPal);
```

Parameter	Meaning
<code>x0</code>	X-position of the upper left corner of the bitmap in the display.
<code>y0</code>	Y-position of the upper left corner of the bitmap in the display.
<code>Xsize</code>	Number of pixels in horizontal direction. Valid range: 1... 255.
<code>Ysize</code>	Number of pixels in vertical direction. Valid range: 1... 255.
<code>XMUL</code>	Scale factor of X-direction.
<code>YMul</code>	Scale factor of Y-direction.
<code>BitsPerPixel</code>	Number of bits per pixel.
<code>BytesPerLine</code>	Number of bytes per line of the image.
<code>pData</code>	Pointer to the actual image, the data that defines what the bitmap looks like.
<code>pPal</code>	Pointer to a <code>GUI_LOGPALETTE</code> structure.

## GUI\_DrawBitmapMag

### Description

This routine makes it possible to scale a bitmap on the display.

### Prototype

```
void GUI_DrawBitmapMag(const GUI_BITMAP* pBM,
```

```
int x0, int y0,
int XMul, int YMul);
```

Parameter	Meaning
<a href="#">pBM</a>	Pointer to the BITMAP to display.
<a href="#">x0</a>	X-position of the upper left corner of the bitmap in the display.
<a href="#">y0</a>	Y-position of the upper left corner of the bitmap in the display.
<a href="#">XMul</a>	Scale factor of X-direction.
<a href="#">YMul</a>	Scale factor of Y-direction.

## GUI\_DrawStreamedBitmap

### Description

This routine draws a bitmap from a data bitmap data stream.

### Prototype

```
void GUI_DrawStreamedBitmap(const GUI_BITMAP_STREAM *pBMH,
                           int x,
                           int y);
```

Parameter	Meaning
<a href="#">pBMH</a>	Pointer to the data stream.
<a href="#">x</a>	X-position of the upper left corner of the bitmap in the display.
<a href="#">y</a>	Y-position of the upper left corner of the bitmap in the display.

### Additional information

You can use the BitmapConverter, described in a later chapter, to create bitmap data streams. The format of these streams is not the same as the format of a BMP-file.

## 6.5 Drawing lines

The most frequently used drawing routines are those that draw a line from one point to another.

### GUI\_DrawHLine

#### Description

Draws a horizontal line one pixel thick from a specified starting point to a specified endpoint in the current window.

#### Prototype

```
void GUI_DrawHLine(int y, int x0, int x1);
```

Parameter	Meaning
<a href="#">y</a>	Y-position.
<a href="#">x0</a>	X-starting position.
<a href="#">x1</a>	X-end position.

### Additional Information

With most LCD controllers, this routine is executed very quickly because multiple pixels can be set at once and no calculations are needed. If it is clear that horizontal lines are to be drawn, this routine executes faster than the `GUI_DrawLine` routine.

## GUI\_DrawLine

### Description

Draws a line from a specified starting point to a specified endpoint in the current window.

### Prototype

```
void GUI_DrawLine(int x0, int y0, int x1, int y1);
```

Parameter	Meaning
<code>x0</code>	X-starting position.
<code>y0</code>	Y-starting position.
<code>x1</code>	X-end position.
<code>y1</code>	Y-end position.

### Additional Information

If part of the line is not visible because it is not in the current window, or if part of the current window is not visible, this part is not drawn due to clipping.

## GUI\_DrawLineRel

### Description

Draws a line from the current (X,Y) position to an endpoint specified by X-distance and Y-distance in the current window.

### Prototype

```
void GUI_DrawLineRel(int dx, int dy);
```

Parameter	Meaning
<code>dx</code>	Distance in X-direction to end of line to draw.
<code>dy</code>	Distance in Y-direction end of line to draw.

## GUI\_DrawLineTo

### Description

Draws a line from the current (X,Y) position to an endpoint specified by X- and Y-coordinates in the current window.

### Prototype

```
void GUI_DrawLineTo(int x, int y);
```

Parameter	Meaning
<code>x</code>	X-end position.
<code>y</code>	Y-end position.

## GUI\_DrawPolyLine

### Description

Connects a predefined list of points with lines in the current window.

### Prototype

```
void GUI_DrawPolyLine(const GUI_POINT* pPoint, int NumPoints, int x, int y);
```

Parameter	Meaning
pPoint	Pointer to the polyline to display.
NumPoints	Number of points specified in the list of points.
x	X-position of origin.
y	Y-position of origin.

### Additional Information

The starting point and endpoint of the polyline need not be identical.

## GUI\_DrawVLine

### Description

Draws a vertical line one pixel thick from a specified starting point to a specified end-point in the current window.

### Prototype

```
void GUI_DrawVLine(int x, int y0, int y1);
```

Parameter	Meaning
x	X-position.
y0	Y-starting position.
y1	Y-end position.

### Additional Information

If `y1 < y0`, nothing will be displayed.  
With most LCD controllers, this routine is executed very quickly because multiple pixels can be set at once and no calculations are needed. If it is clear that vertical lines are to be drawn, this routine executes faster than the `GUI_DrawLine` routine.

# 6.6 Drawing polygons

The polygon drawing routines can be helpful when drawing vectorized symbols.

## GUI\_DrawPolygon

### Description

Draws the outline of a polygon defined by a list of points in the current window.

## Prototype

```
void GUI_DrawPolygon(const GUI_POINT* pPoint, int NumPoints, int x, int y);
```

Parameter	Meaning
<code>pPoint</code>	Pointer to the polygon to display.
<code>NumPoints</code>	Number of points specified in the list of points.
<code>x</code>	X-position of origin.
<code>y</code>	Y-position of origin.

## Additional Information

The polyline drawn is automatically closed by connecting the endpoint to the starting point.

## GUI\_EnlargePolygon

### Description

Enlarges a polygon on all sides by a specified length in pixels.

### Prototype

```
void GUI_EnlargePolygon(GUI_POINT* pDest,
                       const GUI_POINT* pSrc,
                       int NumPoints,
                       int Len);
```

Parameter	Meaning
<code>pDest</code>	Pointer to the destination polygon.
<code>pSrc</code>	Pointer to the source polygon.
<code>NumPoints</code>	Number of points specified in the list of points.
<code>Len</code>	Length (in pixels) by which to enlarge the polygon.

## Additional Information

Make sure the destination array of points is equal to or larger than the source array.

### Example

```
#define countof(Array) (sizeof(Array) / sizeof(Array[0]))

const GUI_POINT aPoints[] = {
    { 0, 20},
    { 40, 20},
    { 20, 0}
};

GUI_POINT aEnlargedPoints[countof(aPoints)];

void Sample(void) {
    int i;
    GUI_Clear();
    GUI_SetDrawMode(GUI_DM_XOR);
    GUI_FillPolygon(aPoints, countof(aPoints), 140, 110);
    for (i = 1; i < 10; i++) {
        GUI_EnlargePolygon(aEnlargedPoints, aPoints, countof(aPoints), i * 5);
        GUI_FillPolygon(aEnlargedPoints, countof(aPoints), 140, 110);
    }
}
```

Screen shot for preceeding example



GUI\_FillPolygon

Description

Draws a filled polygon defined by a list of points in the current window.

Prototype

```
void GUI_FillPolygon(const GUI_POINT* pPoint, int NumPoints, int x, int y);
```

Parameter	Meaning
pPoint	Pointer to the polygon to display and to fill.
NumPoints	Number of points specified in the list of points.
x	X-position of origin.
y	Y-position of origin.

Additional Information

The polyline drawn is automatically closed by connecting the endpoint to the starting point. The endpoint must not touch the outline of the polygon.

GUI\_MagnifyPolygon

Description

Magnifies a polygon by a specified factor.

Prototype

```
void GUI_MagnifyPolygon(GUI_POINT* pDest,
                        const GUI_POINT* pSrc,
                        int NumPoints,
                        int Mag);
```

Parameter	Meaning
pDest	Pointer to the destination polygon.
pSrc	Pointer to the source polygon.
NumPoints	Number of points specified in the list of points.
Mag	Factor used to magnify the polygon.

## Additional Information

Make sure the destination array of points is equal to or larger than the source array. Note the difference between enlarging and magnifying a polygon. Whereas setting the parameter `Len` to 1 will enlarge the polygon by one pixel on all sides, setting the parameter `Mag` to 1 will have no effect.

## Example

```
#define countof(Array) (sizeof(Array) / sizeof(Array[0]))

const GUI_POINT aPoints[] = {
    { 0, 20},
    { 40, 20},
    { 20, 0}
};

GUI_POINT aMagnifiedPoints[countof(aPoints)];

void Sample(void) {
    int Mag, y = 0, Count = 4;
    GUI_Clear();
    GUI_SetColor(GUI_GREEN);
    for (Mag = 1; Mag <= 4; Mag *= 2, Count /= 2) {
        int i, x = 0;
        GUI_MagnifyPolygon(aMagnifiedPoints, aPoints, countof(aPoints), Mag);
        for (i = Count; i > 0; i--, x += 40 * Mag) {
            GUI_FillPolygon(aMagnifiedPoints, countof(aPoints), x, y);
        }
        y += 20 * Mag;
    }
}
```

## Screen shot for preceeding example



## GUI\_RotatePolygon

### Description

Rotates a polygon by a specified angle.

### Prototype

```
void GUI_RotatePolygon(GUI_POINT* pDest,
                      const GUI_POINT* pSrc,
                      int NumPoints,
```

float Angle);

Parameter	Meaning
pDest	Pointer to the destination polygon.
pSrc	Pointer to the source polygon.
NumPoints	Number of points specified in the list of points.
Angle	Angle in radian used to rotate the polygon.

Additional Information

Make sure the destination array of points is equal to or larger than the source array.

Example

The following example shows how to draw a polygon. It can be found under Samples\Misc\DrawPolygon.c.

```

/*****
 *                               Micrium Inc.
 *                               Empowering embedded systems
 *                               uC/GUI sample code
 *****/

-----
File      : DrawPolygon.c
Purpose   : Example for drawing a polygon
-----
*/

#include "gui.h"

/*****
 *                               The points of the arrow
 *****/

static const GUI_POINT aPointArrow[] = {
    { 0, -5},
    {-40, -35},
    {-10, -25},
    {-10, -85},
    { 10, -85},
    { 10, -25},
    { 40, -35},
};

/*****
 *                               Draws a polygon
 *****/

static void DrawPolygon(void) {
    int Cnt =0;
    GUI_SetBkColor(GUI_WHITE);
    GUI_Clear();
    GUI_SetFont(&GUI_Font8x16);
    GUI_SetColor(0x0);
    GUI_DispStringAt("Polygons of arbitrary shape ", 0, 0);
    GUI_DispStringAt("in any color", 120, 20);
    GUI_SetColor(GUI_BLUE);
    /* Draw filled polygon */
    GUI_FillPolygon (&aPointArrow[0],7,100,100);
}
```

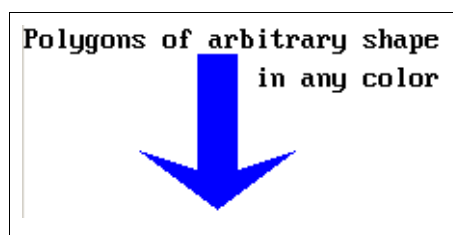
```

/*****
*
*           main
*
*****/

void main(void) {
    GUI_Init();
    DrawPolygon();
    while(1)
        GUI_Delay(100);
}

```

**Screen shot for preceeding example**



## 6.7 Drawing circles

### GUI\_DrawCircle

#### Description

Draws a circle of specified dimensions at a specified position in the current window.

#### Prototype

```
void GUI_DrawCircle(int x0, int y0, int r);
```

Parameter	Meaning
<code>x0</code>	X-position of the center of the circle in pixels of the client window.
<code>y0</code>	Y-position of the center of the circle in pixels of the client window.
<code>r</code>	Radius of the circle (half the diameter). Minimum: 0 (will result in a point). Maximum: 180.

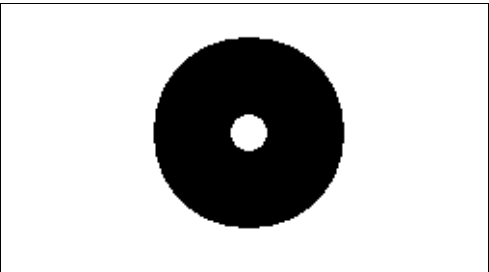
#### Additional Information

This routine cannot handle a radius in excess of 180 because it uses integer calculations that would otherwise produce an overflow. However, for most embedded applications this is not a problem since a circle with diameter 360 is larger than the display anyhow.

**Example**

```
// Draw concentric circles
void ShowCircles(void) {
    int i;
    for (i=10; i<50; i++)
        GUI_DrawCircle(120,60,i);
}
```

**Screen shot for preceeding example**



**GUI\_FillCircle**

**Description**

Draws a filled circle of specified dimensions at a specified position in the current window.

**Prototype**

```
void GUI_FillCircle(int x0, int y0, int r);
```

Parameter	Meaning
x0	X-position of the center of the circle in pixels of the client window.
y0	Y-position of the center of the circle in pixels of the client window.
r	Radius of the circle (half the diameter). Minimum: 0 (will result in a point). Maximum: 180.

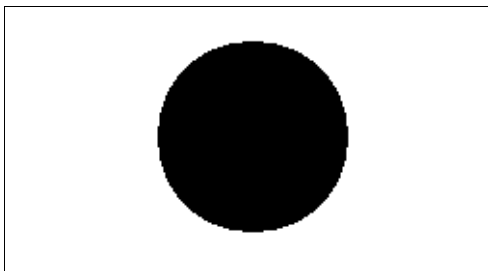
**Additional Information**

This routine cannot handle a radius in excess of 180.

**Example**

```
GUI_FillCircle(120,60,50);
```

## Screen shot for preceeding example



## 6.8 Drawing ellipses

### GUI\_DrawEllipse

#### Description

Draws an ellipse of specified dimensions at a specified position in the current window.

#### Prototype

```
void GUI_DrawEllipse (int x0, int y0, int rx, int ry);
```

Parameter	Meaning
x0	X-position of the center of the circle in pixels of the client window.
y0	Y-position of the center of the circle in pixels of the client window.
rx	X-radius of the ellipse (half the diameter). Minimum: 0, Maximum: 180.
ry	Y-radius of the ellipse (half the diameter). Minimum: 0, Maximum: 180.

#### Additional Information

This routine cannot handle rx/ry parameters in excess of 180 because it uses integer calculations that would otherwise produce an overflow.

#### Example

See the GUI\_FillEllipse example.

### GUI\_FillEllipse

#### Description

Draws a filled ellipse of the given dimensions.

Prototype

```
void GUI_FillEllipse(int x0, int y0, int rx, int ry);
```

Parameter	Meaning
x0	X-position of the center of the circle in pixels of the client window.
y0	Y-position of the center of the circle in pixels of the client window.
rx	X-radius of the ellipse (half the diameter). Minimum: 0, Maximum: 180.
ry	Y-radius of the ellipse (half the diameter). Minimum: 0, Maximum: 180.

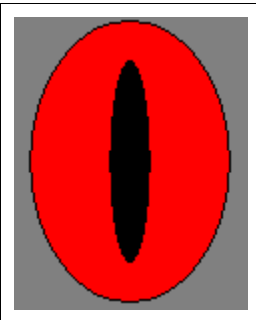
Additional Information

This routine cannot handle a rx/ry parameters in excess of 180.

Example

```
/*
 Demo ellipses
*/
GUI_SetColor(0xff);
GUI_FillEllipse(100, 180, 50, 70);
GUI_SetColor(0x0);
GUI_DrawEllipse(100, 180, 50, 70);
GUI_SetColor(0x000000);
GUI_FillEllipse(100, 180, 10, 50);
```

Screen shot for preceeding example



6.9 Drawing arcs

GUI\_DrawArc

Description

Draws an arc of specified dimensions at a specified position in the current window. An arc is a section of the outline of a circle.

## Prototype

```
void GL_DrawArc (int xCenter, int yCenter, int rx, int ry, int a0, int a1);
```

Parameter	Meaning
<a href="#">xCenter</a>	Horizontal position of the center in pixels of the client window.
<a href="#">yCenter</a>	Vertical position of the center in pixels of the client window.
<a href="#">rx</a>	X-radius (pixels).
<a href="#">ry</a>	Y-radius (pixels).
<a href="#">a0</a>	Starting angle (degrees).
<a href="#">a1</a>	Ending angle (degrees).

## Limitations

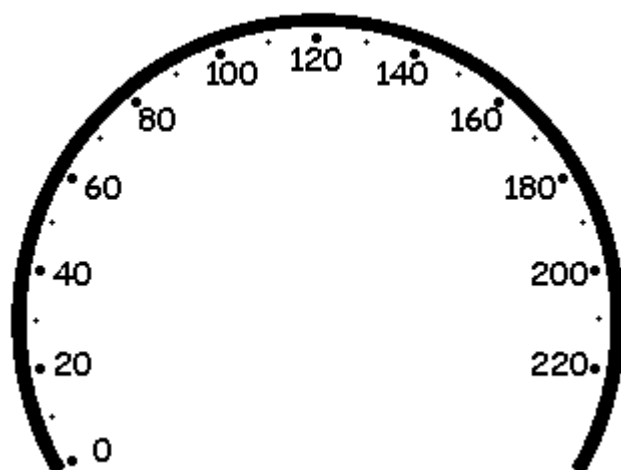
Currently the ry parameter is not used, and the rx parameter is used instead.

## Additional Information

GUI\_DrawArc uses the floating-point library. It cannot handle rx/ry parameters in excess of 180 because it uses integer calculations that would otherwise produce an overflow.

## Example

```
void DrawArcScale(void) {
    int x0 = 160;
    int y0 = 180;
    int i;
    char ac[4];
    GUI_SetBkColor(GUI_WHITE);
    GUI_Clear();
    GUI_SetPenSize( 5 );
    GUI_SetTextMode(GUI_TM_TRANS);
    GUI_SetFont(&GUI_FontComic18B_ASCII);
    GUI_SetColor( GUI_BLACK );
    GUI_DrawArc( x0,y0,150, 150,-30, 210 );
    GUI_Delay(1000);
    for (i=0; i<= 23; i++) {
        float a = (-30+i*10)*3.1415926/180;
        int x = -141*cos(a)+x0;
        int y = -141*sin(a)+y0;
        if (i%2 == 0)
            GUI_SetPenSize( 5 );
        else
            GUI_SetPenSize( 4 );
        GUI_DrawPoint(x,y);
        if (i%2 == 0) {
            x = -123*cos(a)+x0;
            y = -130*sin(a)+y0;
            sprintf(ac, "%d", 10*i);
            GUI_SetTextAlign(GUI_TA_VCENTER);
            GUI_DispatchStringHCenterAt(ac,x,y);
        }
    }
}
```

**Screen shot for preceeding example**

# Chapter 7

## Fonts

---

$\mu$ C/GUIThe most common fonts are shipped with  $\mu$ C/GUI as standard fonts. In fact, you will probably find that these fonts are fully sufficient for your application. For detailed information on the individual fonts, please see Chapter 25: "Standard Fonts", which describes all fonts included with  $\mu$ C/GUI and shows all characters as they appear on the display.

$\mu$ C/GUI supports ASCII, ISO 8859-1 and Unicode. Usually,  $\mu$ C/GUI is compiled for 8-bit characters, allowing for a maximum of 256 different character codes out of which the first 32 are reserved as control characters. The characters that are available depends on the selected font.

## 7.1 Available fonts

The current version of  $\mu$ C/GUI supports 4 types of fonts: monospaced bitmap fonts, proportional bitmap fonts, proportional bitmap fonts with 2 bpp built-in antialiasing information and antialiased fonts with 4 bpp built-in antialiasing information. (For more information on antialiased fonts, see Chapter 15: "Antialiasing".)

All fonts linked with your application and declared (in `GUIConf.h`) may be selected. We recommend compiling all available fonts and linking them as library modules or putting all of the font object files in a library which you can link with your application. This way you can be sure that only the fonts which are needed by your application are actually linked. The font converter (described in a separate manual) may be used to create additional fonts.

In order to be able to use a font in your application, the following requirements must be met:

- The font is in a form compatible with  $\mu$ C/GUI as either "C" file, object file or library.
- The font file is linked with your application.
- The font declaration is contained in `GUIConf.h` (this is necessary in order to avoid compiler warnings due to undeclared externals).

### Adding fonts

Once you have linked a font file as described above, declare the linked font as `extern const GUI_FONT`, as shown in the example below:

#### Example

```
extern const GUI_FONT GUI_FontNew;

int main(void) {
    GUI_Init();
    GUI_Clear();
    GUI_SetFont(&GUI_FontNew);
    GUI_DispString("Hello world\n");
    return 0;
}
```

### Selecting the font

$\mu$ C/GUI offers different fonts, one of which is always selected. This selection can be changed by calling the function `GUI_SetFont()`, which selects the font to use for all text output to follow for the current task.

If no font has been selected by your application, the default font is used. This default is configured in `GUIConf.h` and can be changed. You should make sure that the default font is one that you are actually using in your application because the default font will be linked with your application and will therefore use up ROM memory.

#### $\mu$ C/GUICompatibility

Older versions of  $\mu$ C/GUI used a different font concept where fonts were listed in a font table and their position in that table was selected by an integer. This concept was changed due to a lack of flexibility. The new concept is to a large degree compatible with the old one, and the font identifiers (i.e. F6x8) are still available.

## 7.2 Font API

The table below lists the available font-related routines in alphabetical order within their respective categories. Detailed descriptions can be found in the sections that follow.

Routine	Explanation
Selection of a font	
<a href="#">GUI_GetFont()</a>	Return a pointer to the currently selected font.
<a href="#">GUI_SetFont()</a>	Set the current font.
Font-related functions	
<a href="#">GUI_GetCharDistX()</a>	Return the width in pixels (X-size) of a specified character in the current font.
<a href="#">GUI_GetFontDistY()</a>	Return the Y-spacing of the current font.
<a href="#">GUI_GetFontInfo()</a>	Return a structure containing font information.
<a href="#">GUI_GetFontSizeY()</a>	Return the height in pixels (Y-size) of the current font.
<a href="#">GUI_GetStringDistX()</a>	Return the X-size of a text using the current font.
<a href="#">GUI_GetYDistOfFont()</a>	Return the Y-spacing of a particular font.
<a href="#">GUI_GetYSizeOfFont()</a>	Return the Y-size of a particular font.
<a href="#">GUI_IsInFont()</a>	Evaluate whether a specified character is in a particular font.

## 7.3 Selection of a font

### GUI\_GetFont()

#### Description

Returns a pointer to the currently selected font.

#### Prototype

```
const GUI_FONT * GUI_GetFont(void)
```

### GUI\_SetFont()

#### Description

Sets the font to be used for text output.

#### Prototype

```
const GUI_FONT * GUI_SetFont(const GUI_FONT * pNewFont);
```

Parameter	Meaning
<a href="#">pFont</a>	Pointer to the font to be selected and used.

#### Return value

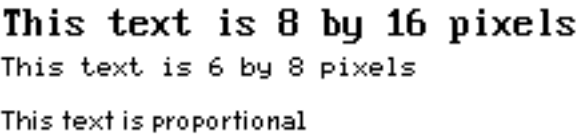
Returns a pointer to the previously selected font so that it may be restored at a later point.

#### Examples

Displays sample text in 3 different sizes, restoring the former font afterwards:

```
void DispText(void) {
    const GUI_FONT GUI_FLASH* OldFont=GUI_SetFont(&GUI_Font8x16);
    GUI_DisStringAt("This text is 8 by 16 pixels",0,0);
    GUI_SetFont(&GUI_Font6x8);
    GUI_DisStringAt("This text is 6 by 8 pixels",0,20);
    GUI_SetFont(&GUI_Font8);
    GUI_DisStringAt("This text is proportional",0,40);
    GUI_SetFont(OldFont);           // Restore font
}
```

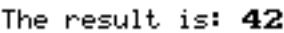
Screen shot of above example:



Displays text and value in different fonts:

```
GUI_SetFont(&GUI_Font6x8);
GUI_DisString("The result is: "); // Disp text
GUI_SetFont(&GUI_Font8x8);
GUI_DisDec(42,2); // Disp value
```

Screen shot of above example:



# 7.4 Font-related functions

## GUI\_GetCharDistX()

### Description

Returns the width in pixels (X-size) used to display a specified character in the currently selected font.

### Prototype

```
int GUI_GetCharDistX(U16 c);
```

Parameter	Meaning
c	Character to calculate width from.

## GUI\_GetFontDistY()

### Description

Returns the Y-spacing of the currently selected font.

### Prototype

```
int GUI_GetFontDistY(void);
```

### Additionnal information

The Y-spacing is the vertical distance in pixels between two adjacent lines of text. The returned value is the YDist value of the entry for the currently selected font. The returned value is valid for both proportional and monospaced fonts.

## GUI\_GetFontInfo()

### Description

Calculates a pointer to a GUI\_FONTINFO structure of a particular font.

### Prototype

```
void GUI_GetFontInfo(const GUI_FONT* pFont, GUI_FONTINFO* pfi);
```

Parameter	Meaning
<code>pFont</code>	Pointer to the font.
<code>pfi</code>	Pointer to a GUI_FONTINFO structure.

### Additional information

The definition of the GUI\_FONTINFO structure is as follows:

```
typedef struct {
    U16 Flags;
} GUI_FONTINFO;
```

The member variable flags can take the following values:

```
GUI_FONTINFO_FLAG_PROP
GUI_FONTINFO_FLAG_MONO
GUI_FONTINFO_FLAG_AA
GUI_FONTINFO_FLAG_AA2
GUI_FONTINFO_FLAG_AA4
```

### Example

Gets the info of GUI\_Font6x8. After the calculation, FontInfo.Flags contains the flag GUI\_FONTINFO\_FLAG\_MONO.

```
GUI_FONTINFO FontInfo;
GUI_GetFontInfo(&GUI_Font6x8, &FontInfo);
```

## GUI\_GetFontSizeY()

### Description

Returns the height in pixels (Y-size) of the currently selected font.

### Prototype

```
int GUI_GetFontSizeY(void);
```

### Additional information

The returned value is the YSize value of the entry for the currently selected font. This value is less than or equal to the Y-spacing returned by the function GUI\_GetFontDistY().

The returned value is valid for both proportional and monospaced fonts.

## GUI\_GetStringDistX()

### Description

Returns the X-size used to display a specified string in the currently selected font.

### Prototype

```
int GUI_GetStringDistX(const char GUI_FAR *s);
```

Parameter	Meaning
<code>s</code>	Pointer to the string.

## GUI\_GetYDistOfFont()

### Description

Returns the Y-spacing of a particular font.

### Prototype

```
int GUI_GetYDistOfFont(const GUI_FONT* pFont);
```

Parameter	Meaning
<a href="#">pFont</a>	Pointer to the font.

### Additional information

(see `GUI_GetFontDistY()`)

## GUI\_GetYSizeOfFont()

### Description

Returns the Y-size of a particular font.

### Prototype

```
int GUI_GetYSizeOfFont(const GUI_FONT* pFont);
```

Parameter	Meaning
<a href="#">pFont</a>	Pointer to the font.

### Additional information

(see `GUI_GetFontSizeY()`)

## GUI\_IsInFont()

### Description

Evaluates whether or not a particular font contains a specified character.

### Prototype

```
char GUI_IsInFont(const GUI_FONT*pFont, U16 c);
```

Parameter	Meaning
<a href="#">pFont</a>	Pointer to the font.
<a href="#">c</a>	Character to be searched for.

### Additional information

If the pointer [pFont](#) is set to 0, the currently selected font is used.

### Example

Evaluates whether the font `GUI_FontD32` contains an "X":

```
if (GUI_IsInFont(&GUI_FontD32, 'X') == 0) {  
    GUI_DisString("GUI_FontD32 does not contains 'X'");  
}
```

## 7.5 Character sets

### ASCII

µC/GUI supports the full set of ASCII characters. These are the following 96 characters from 32 to 127:

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2x		!		"#	\$	%	&		'(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x		`a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{		}	~	

Unfortunately, as ASCII stands for American Standard Code for Information Interchange, it is designed for American needs. It does not include any of the special characters used in European languages, such as Ä, Ö, Ü, á, à, and others. There is no single standard for these "European extensions" of the ASCII set of characters; several different ones exist. The one used on the Internet and by most Windows programs is ISO 8859-1, a superset of the ASCII set of characters.

### ISO 8859-1 Western Latin character set

µC/GUI supports the ISO 8859-1, which defines characters as listed below:

Code	Description	Char
160	non-breaking space	
161	inverted exclamation	¡
162	cent sign	¢
163	pound sterling	£
164	general currency sign	¤
165	yen sign	¥
166	broken vertical bar	¦
167	section sign	§
168	umlaut (dieresis)	¨
169	copyright	©
170	feminine ordinal	ª
171	left angle quote, guillemotleft	«
172	not sign	¬
173	soft hyphen	–
174	registered trademark	®
175	macron accent	¯
176	degree sign	°
177	plus or minus	±
178	superscript two	²
179	superscript three	³
180	acute accent	´
181	micro sign	µ
182	paragraph sign	¶
183	middle dot	·

Code	Description	Char
184	cedilla	¸
185	superscript one	¹
186	masculine ordinal	º
187	right angle quote, guillemot right	»
188	fraction one-fourth	¼
189	fraction one-half	½
190	fraction three-fourth	¾
191	inverted question mark	¿
192	capital A, grave accent	À
193	capital A, acute accent	Á
194	capital A, circumflex accent	Â
195	capital A, tilde	Ã
196	capital A, dieresis or umlaut mark	Ä
197	capital A, ring	Å
198	capital A, diphthong (ligature)	Æ
199	capital C, cedilla	Ç
200	capital E, grave accent	È
201	capital E, acute accent	É
202	capital E, circumflex accent	Ê
203	capital E, dieresis or umlaut mark	Ë
204	capital I, grave accent	Ì
205	capital I, acute accent	Í
206	capital I, circumflex accent	Î
207	capital I, dieresis or umlaut mark	Ï
208	Eth, Icelandic	Ð
209	N, tilde	Ñ
210	capital O, grave accent	Ò
211	capital O, acute accent	Ó
212	capital O, circumflex accent	Ô
213	capital O, tilde	Õ
214	capital O, dieresis or umlaut mark	Ö
215	multiply sign	×
216	capital O, slash	Ø
217	capital U, grave accent	Ù
218	capital U, acute accent	Ú
219	capital U, circumflex accent	Û
220	capital U, dieresis or umlaut mark	Ü
221	capital Y, acute accent	Ý
222	THORN, Icelandic	Þ
223	sharp s, German (s-z ligature)	ß
224	small a, grave accent	à
225	small a, acute accent	á
226	small a, circumflex accent	â
227	small a, tilde	ã
228	small a, dieresis or umlaut mark	ä
229	small a, ring	å
230	small ae diphthong (ligature)	æ
231	cedilla	ç
232	small e, grave accent	è
233	small e, acute accent	é
234	small e, circumflex accent	ê
235	small e, dieresis or umlaut mark	ë
236	small i, grave accent	ì

Code	Description	Char
237	small i, acute accent	í
238	small i, circumflex accent	î
239	small i, dieresis or umlaut mark	ï
240	small eth, Icelandic	ð
241	small n, tilde	ñ
242	small o, grave accent	ò
243	small o, acute accent	ó
244	small o, circumflex accent	ô
245	small o, tilde	õ
246	small o, dieresis or umlaut mark	ö
247	division sign	÷
248	small o, slash	ø
249	small u, grave accent	ù
250	small u, acute accent	ú
251	small u, circumflex accent	û
252	small u, dieresis or umlaut mark	ü
253	small y, acute accent	ý
254	small thorn, Icelandic	þ
255	small y, dieresis or umlaut mark	ÿ

## Unicode

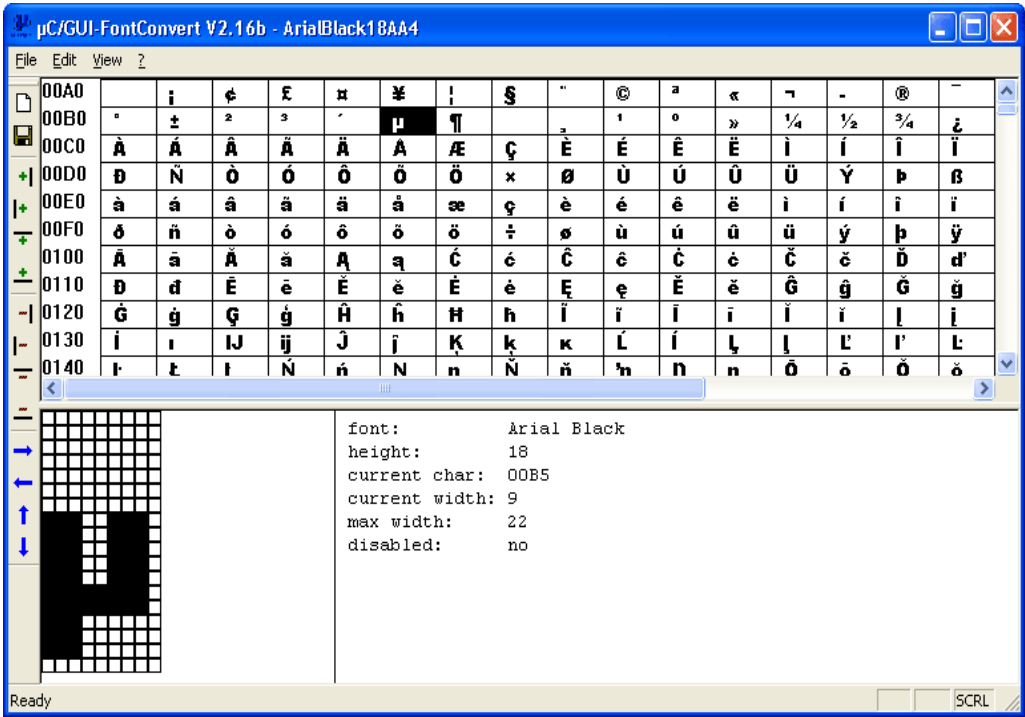
Unicode is the ultimate in character coding. It is an international standard based on ASCII and ISO 8859-1. Contrary to ASCII, UNICODE requires 16-bit characters because all characters have their own code. Currently, more than 30,000 different characters are defined. However, not all of the character images are defined in  $\mu$ C/GUI. It is the responsibility of the user to define these additional characters. Please contact Micrium Inc. or your distributor, as we may already have the character set that you need.

## 7.6 Font converter

Fonts which can be used with  $\mu$ C/GUI must be defined as `GUI_FONT` structures in "C". The structures -- or rather the font data which is referenced by these structures -- can be rather large. It is very time-consuming and inefficient to generate these fonts manually. We therefore recommend using the font converter, which automatically generates "C" files from fonts.

The font converter is a simple Windows program. You need only to load an installed Windows font into the program, edit it if you want or have to, and save it as a "C" file. The "C" file may then be compiled, allowing the font to be shown on your display with  $\mu$ C/GUI on demand.

The character codes 0x00 - 0x1F and 0x80 - 0x9F are disabled by default. The following is a sample screen shot of the font converter with a font loaded:



The font converter is described in a separate documentation which can be obtained by request from Micrium Inc. ([info@micrium.com](mailto:info@micrium.com)).

# Chapter 8

## Bitmap Converter

---

Bitmaps which can be used with  $\mu$ C/GUI are normally defined as `GUI_BITMAP` structures in C. The structures -- or rather the picture data which is referenced by these structures -- can be quite large. It is time-consuming and inefficient to generate these bitmaps manually, especially if you are dealing with sizable images and multiple shades of gray or colors. We therefore recommend using the bitmap converter.

The bitmap converter is a Windows program which is easy to use. Simply load a bitmap (in the form of a `.bmp` file) into the program, convert it if you want or have to, and save it as a C file, which can then be used by  $\mu$ C/GUI and shown on your display.

## 8.1 Introduction

The bitmap converter is intended as a tool to convert bitmaps from a PC format to a C file. It also enables color conversion, so that the resulting C code is not unnecessarily large. You would typically reduce the pixel depth in order to reduce memory consumption. The bitmap converter shows the converted image.

A limited number of simple functions can be performed with the bitmap converter, including flipping the bitmap horizontally or vertically, rotating it, and inverting the bitmap indices or colors (these features can be found under the `Image` menu). Any further modifications to an image must be made in a bitmap manipulation program such as Adobe Photoshop or Corel Photopaint. It usually makes the most sense to perform any image modifications in such a program, using the bitmap converter for converting purposes only.

## 8.2 Supported input formats

An image must first be loaded into the bitmap converter as a `.bmp` file, either directly or via the clipboard. The following `.bmp` files may be loaded:

- 1,4, or 8 bits per pixel (bpp) with palette;
- 24 bpp without palette (RGB/full-color mode);
- RLE4 and RLE8.

Other graphic formats may be copied to the clipboard, changed to a `.bmp` file, and then loaded to the bitmap converter.

## 8.3 Supported output formats

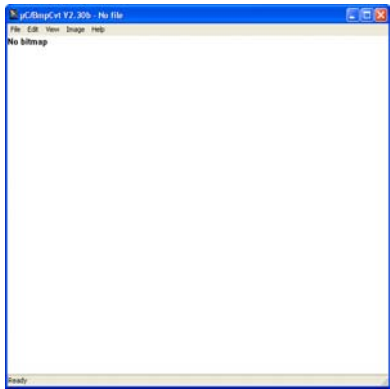
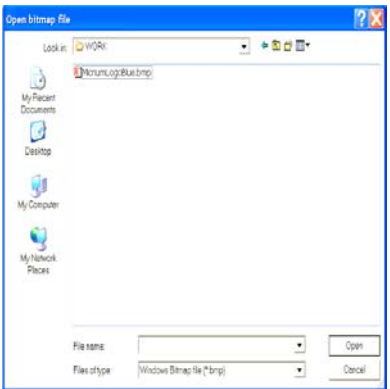
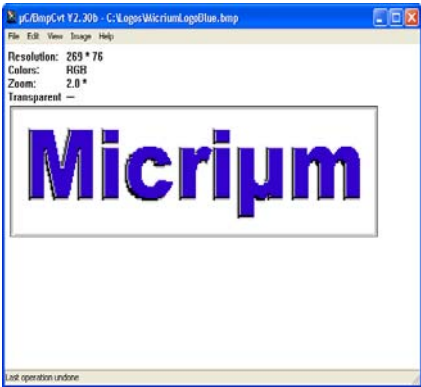
The converted bitmap may be saved as a `.bmp` file (which can be reloaded and used or loaded into other bitmap manipulation programs) or as a C file. A C file will serve as an input file for your C compiler. It may contain a palette (device-independent bitmap, or DIB) or be saved without (device-dependent bitmap, or DDB). DIBs are recommended, as they will display correctly on any LCD; a DDB will only display correctly on an LCD which uses the same palette as the bitmap.

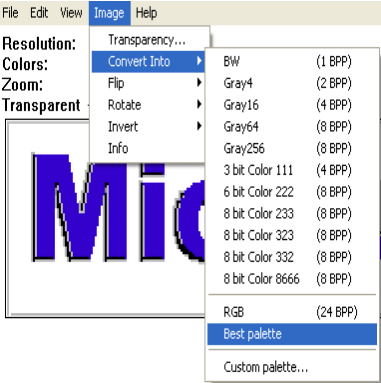
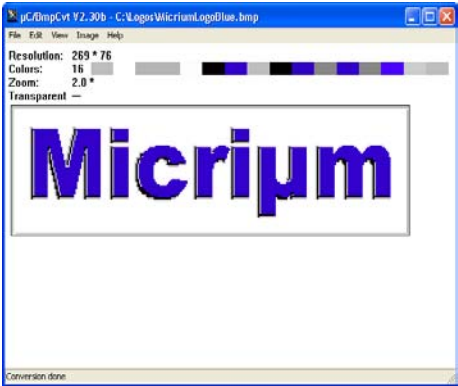
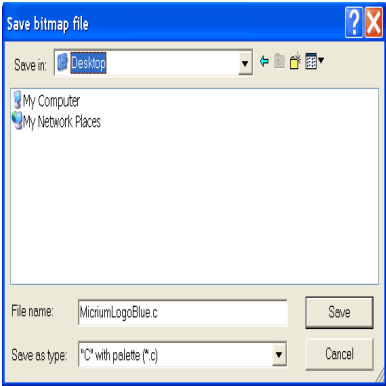
## 8.4 Generating C files from bitmaps

The main function of the bitmap converter is to convert PC-formatted bitmaps into C files which can be used by  $\mu$ C/GUI. Before doing so, however, it is often desirable to modify the color palette of an image so that the generated C file is not excessively large. With full-color bitmaps, it will be necessary to convert the image into a palette format, as the bitmap converter cannot generate C files from bitmaps in RGB mode.

C files may be generated as "C with palette", "C without palette", "C with palette, compressed" or "C without palette, compressed". For more information on compressed files, see the section "Compressed bitmaps" as well as the example at the end of the chapter.

The basic procedure for using the bitmap converter is illustrated step by step in the table below:

Procedure	Screen shot
<p>Step 1: Start the application.</p> <p>The bitmap converter is opened as an empty window.</p>	
<p>Step 2: Load a bitmap into the bitmap converter.</p> <p>Choose File/Open.</p> <p>Locate the document you want to open and click Open (must be a .bmp file).</p> <p>In this example, the file MicriumLogo.bmp is chosen.</p>	
<p>Step 3: The bitmap converter displays the loaded (source) bitmap.</p> <p>In this example, the source bitmap is in full-color (RGB) mode. It therefore must be converted to a palette format before a C file can be generated.</p>	

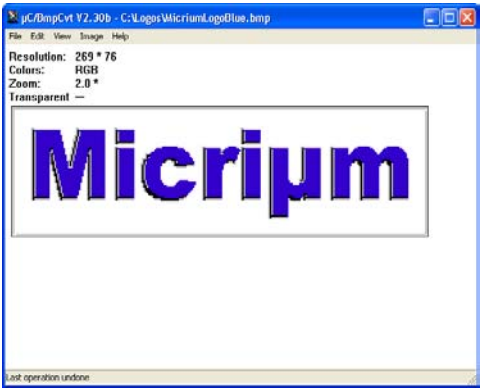
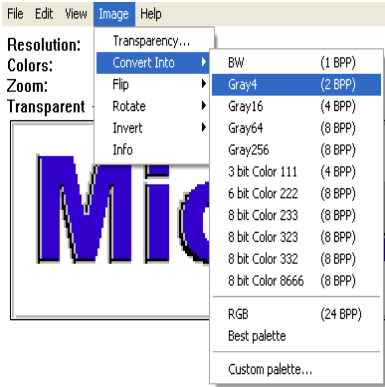
Procedure	Screen shot
<p>Step 4: Convert the image if necessary.</p> <p>Choose Image/Convert Into. Select the desired palette. In this example, the option Best palette is chosen.</p>	
<p>Step 5: The bitmap converter displays the converted bitmap.</p> <p>In this example, the image is unchanged in terms of appearance, but uses less memory since a palette of only 16 colors is used instead of the full-color mode. These 16 colors are the only ones actually required to display this particular image.</p>	
<p>Step 6: Save the bitmap as a C file.</p> <p>Choose File/Save. Select a destination and a name for the C file. Select the file type. In this example, the file is saved as "C with palette." Click Save. The bitmap converter will create a separate file in the specified destination, containing the C source code for the bitmap.</p>	


8.5 Color conversion

The primary reason for converting the color format of a bitmap is to reduce memory consumption and therefore generate a more efficient C file.

The most common way of doing this is by using the option `Best palette` as in the above example, which customizes the palette of a particular bitmap to include only the colors which are used in the image. It is especially useful with full-color bitmaps in order to make the palette as small as possible while still fully supporting the image. Once a bitmap file has been opened in the bitmap converter, simply select `Image/Convert Into/Best palette` from the menu.

For certain applications, it may be more efficient to use a fixed color palette, chosen from the menu under `Image/Convert Into`. For example, suppose a bitmap in full-color mode is to be shown on a display which supports only four grayscales. It would be a waste of memory to keep the image in the original format, since it would only appear as four grayscales on the display. The procedure for conversion would be as follows:

Procedure	Screen shot
<p>The bitmap converter displays the loaded (source) bitmap, which is the same as in the previous example (full-color mode).</p>	 <p>The screenshot shows the µC/BmpCvt V2.20b application window. The title bar reads 'µC/BmpCvt V2.20b - C:\Lagos\Micrium\logoBlue.bmp'. The menu bar includes File, Edit, View, Image, and Help. On the left, status information is displayed: Resolution: 255 * 76, Colors: RGB, Zoom: 2.0 *, and Transparent: —. The main canvas displays a blue logo with the word 'Micrium' in a stylized font.</p>
<p>Choose <code>Image/Convert Into/Gray4</code>.</p>	 <p>This screenshot shows the same application window with the 'Image' menu open. The 'Convert Into' option is selected, which has opened a submenu. In this submenu, 'Gray4 (2 BPP)' is highlighted. Other options visible in the submenu include BW (1 BPP), Gray16 (4 BPP), Gray64 (8 BPP), Gray256 (8 BPP), 3 bit Color 111 (4 BPP), 6 bit Color 222 (8 BPP), 8 bit Color 233 (8 BPP), 8 bit Color 323 (8 BPP), 8 bit Color 332 (8 BPP), 8 bit Color 8666 (8 BPP), RGB (24 BPP), Best palette, and Custom palette...</p>

Procedure	Screen shot
<p>The bitmap converter displays the converted bit-map.</p> <p>In this example, the image uses less memory since a palette of only 4 grayscales is used instead of the full-color mode. If the target display supports only 4 grayscales, there is no use in having a higher pixel depth as it would only waste memory.</p>	 The screenshot shows a window titled "µC/GuiPvt V2.30b - C:\Logos\MicriumLogoBlue.bmp". The menu bar includes File, Edit, View, Image, and Help. Below the menu bar, the following properties are listed: Resolution: 269 * 76, Colors: 4 (with a grayscale bar), Zoom: 2.0 *, and Transparent: —. The main area displays the word "Micrium" in a bold, stylized font. At the bottom left of the window, a status bar says "Conversion done".

## 8.6 Compressed bitmaps

The bitmap converter and µC/GUI support run-length encoding (RLE) compression of bitmaps in the resulting source code files. The RLE compression method works most efficiently if your bitmap contains many sequences of equal-colored pixels. An efficiently compressed bitmap will save a significant amount of space. However, compression is not recommended for photographic images since they do not normally have sequences of identical pixels. It should also be noted that a compressed image may take slightly longer to display, since it must first be decompressed.

If you want to save a bitmap using RLE compression, you can do so by selecting one of the compressed output formats when saving as a C file: "C with palette, compressed" or "C without palette, compressed". There are no special functions needed for displaying compressed bitmaps; it works in the same way as displaying uncompressed bitmaps.

## 8.7 Using a custom palette

Under certain circumstances it may be desirable to use a custom palette for conversions. In these cases (usually only if you have a color display using a special custom palette and you would like to use the same palette for bitmaps) a custom palette may be used. In the menu, you would select `Image/Convert Into/Custom palette`.

### File format for custom palette

Custom palette files are simple files defining the available colors for conversion. They contain the following:

- Header (8 bytes).
- NumColors (U32, 4 bytes).
- 0 (4 bytes).
- U32 Colors[NumColors] (NumColors\*4 bytes, type GUI\_COLOR).

Total file size is therefore:  $16 + (\text{NumColors} * 4)$  bytes. A custom palette file with 8 colors would be  $16 + (8 * 4) = 48$  bytes. At this point, a binary editor must be used in order to create such a file.

The maximum number of colors supported is 256; the minimum is 2.

## Custom palette sample file

This sample file would define a palette containing 2 colors -- red and white:

```
0000: 65 6d 57 69 6e 50 61 6c 02 00 00 00 00 00 00 00
0010: ff 00 00 00 ff ff ff 00
```

The 8 headers make up the first eight bytes of the first line. The U32 is stored LSB first (big endian) and represents the next four bytes, followed by the four 0 bytes. Colors are stored 1 byte per color, where the 4th byte is 0 as follows: RRGGBB00. The second line of code therefore defines the two colors used in this sample.

## 8.8 Command line usage

It is also possible to work with the bitmap converter using the command prompt. All conversion functions available in the bitmap converter menu are available as commands, and any number of functions may be performed on a bitmap in one command line. Commands are entered using the following format:

```
BmpCvt <filename>.bmp <-command>
```

(If more than one command is used, one space is typed between each.)

For example, a bitmap with the name "MicriumLogo" is converted into Best palette format and saved as a C file named "logo2" all at once by entering the following at the command prompt:

```
BmpCvt MicriumLogo200.bmp -convertintobestpalette -saveaslogo2,1 -exit
```

Note that while the file to be loaded into the bitmap converter always includes its .bmp extension, no file extension is written in the -saveas command. An integer is used instead to specify the desired file type. The number 1 in the -saveas command above designates "C with palette". The -exit command automatically closes the program upon completion. See the table below for more information.

### Valid command line options

The following table lists all permitted bitmap converter commands. It can also be viewed at any time by entering `BmpCvt /?` at the command prompt.

Command	Explanation
<code>-convertintobw</code>	Convert to BW.
<code>-convertintogray4</code>	Convert to Gray4.
<code>-convertintogray16</code>	Convert to Gray16.
<code>-convertintogray64</code>	Convert to Gray64.
<code>-convertintogray256</code>	Convert to Gray256.
<code>-convertinto111</code>	Convert to 111.
<code>-convertinto222</code>	Convert to 222.
<code>-convertinto233</code>	Convert to 233.

Command	Explanation
<code>-convertinto323</code>	Convert to 323.
<code>-convertinto332</code>	Convert to 332.
<code>-convertinto8666</code>	Convert to 8666.
<code>-convertintoRGB</code>	Convert to RGB.
<code>-convertintobestpalette</code>	Convert to best palette.
<code>-convertintocustompalette&lt;filename&gt;</code>	Convert to a custom palette.
Parameter:	<code>filename</code> : user-specified filename of desired custom palette.
<code>-fliph</code>	Flip image horizontally.
<code>-flipv</code>	Flip image vertically.
<code>-rotate90cw</code>	Rotate image by 90 degrees clockwise.
<code>-rotate90cc</code>	Rotate image by 90 degrees counterclockwise.
<code>-rotate180</code>	Rotate image by 180 degrees.
<code>-invertindices</code>	Invert indices.
<code>-saveas&lt;filename,type&gt;</code>	Save file as filename.
Parameter:	<code>filename</code> : user-specified; does NOT include file extension!
Parameter:	<code>type</code> : must be an integer from 1 to 6 as follows:
	1: C with palette (.c file)
	2: C without palette (.c file)
	3: C with palette, compressed (.c file)
	4: C without palette, compressed (.c file)
	5: stream (.dta file)
	6: Windows Bitmap file (.bmp file)
<code>-exit</code>	Terminate PC program automatically.
<code>-help</code>	Display this box.
<code>-?</code>	Display this box.

## 8.9 Example of a converted bitmap

A typical example for the use of the bitmap converter would be the conversion of your company logo into a C bitmap. Take a look at the sample bitmap pictured:



The bitmap is loaded into the bitmap converter, converted to `Best palette`, and saved as "C with palette". The resulting C source code is displayed below (some data is not shown to conserve space).





*the highest 8 bits (of the 24 bits used) represent the Blue component as follows: 0xBBGGRR*

```

const GUI_COLOR ColorsLogoCompressed[] = {
    0xBFBBBF,0xFFFFF,0xB5B5B5,0x000000
    ,0xFF004C,0xB5002B,0x888888,0xCF0038
    ,0xCFCFCF,0xC0C0C0
};

const GUI_LOGPALETTE PalLogoCompressed = {
    10,/* number of entries */
    0,/* No transparency */
    &ColorsLogoCompressed[0]
};

const unsigned char acLogoCompressed[] = {
    /* RLE: 270 Pixels @ 000,000*/254, 0x00, 16, 0x00,
    /* RLE: 268 Pixels @ 001,001*/254, 0x01, 14, 0x01,
    /* RLE: 001 Pixels @ 000,002*/1, 0x00,
    /* RLE: 267 Pixels @ 001,002*/254, 0x01, 13, 0x01,
    /* ABS: 002 Pixels @ 268,002*/0, 2, 0x20,
    .
    .
    .
    /* ABS: 002 Pixels @ 268,073*/0, 2, 0x20,
    /* RLE: 267 Pixels @ 001,074*/254, 0x01, 13, 0x01,
    /* ABS: 003 Pixels @ 268,074*/0, 3, 0x20, 0x10,
    /* RLE: 267 Pixels @ 002,075*/254, 0x02, 13, 0x02,

    0}; /* 4702 for 20444 pixels */

const GUI_BITMAP bmLogoCompressed = {
    269,/* XSize */
    76,/* YSize */
    135,/* BytesPerLine */
    GUI_COMPRESS_RLE4,/* BitsPerPixel */
    acLogoCompressed, /* Pointer to picture data (indices) */
    &PalLogoCompressed /* Pointer to palette */
    ,GUI_DRAW_RLE4
};

/* *** End of file *** */

```



# Chapter 9

## Colors

---

µC/GUI supports black/white, grayscale (monochrome with different intensities) and color displays. The same user program can be used with any display; only the LCD-configuration needs to be changed. The color management tries to find the closest match for any color that should be displayed.

**Logical colors** are the colors the application deals with. A logical colors is always defined as an RGB value. This is a 24-bit value containing 8 bits per color as follows: 0xBBGGRR. Therefore, white would be 0xFFFFFF, black would be 0x000000, bright red 0xFF.

**Physical colors** are the colors which can actually be displayed by the display. They are specified in the same 24-bit RGB format as logical colors. At run-time, logical colors are mapped to physical colors.

For displays with few colors (such as monochrome displays or 8/16-color LCDs), µC/GUI converts them by using an optimized version of the "least-square deviation search". It compares the color to display (the logical color) with all the available colors that the LCD can actually show (the physical colors) and uses the one that the LCD-metric considers closest.

## 9.1 Predefined colors

In addition to self-defined colors, some standard colors are predefined in  $\mu$ C/GUI, as shown in the following table:

Define	Color	Definition
GUI_BLACK	Black	0x000000
GUI_BLUE	Blue	0xFF0000
GUI_GREEN	Green	0x00FF00
GUI_CYAN	Cyan	0xFFFF00
GUI_RED	Red	0x0000FF
GUI_MAGENTA	Magenta	0x8B008B
GUI_BROWN	Brown	0x2A2AA5
GUI_DARKGRAY	Dark gray	0x404040
GUI_GRAY	Gray	0x808080
GUI_LIGHTGRAY	Light gray	0xD3D3D3
GUI_LIGHTBLUE	Light blue	0xFF8080
GUI_LIGHTGREEN	Light green	0x80FF80
GUI_LIGHTCYAN	Light cyan	0x80FFFF
GUI_LIGHTRED	Light red	0x8080FF
GUI_LIGHTMAGENTA	Light magenta	0xFF80FF
GUI_YELLOW	Yellow	0x00FFFF
GUI_WHITE	White	0xFFFFFFFF

### Example:

```
/* Set background color to magenta */
GUI_SetBkColor(GUI_MAGENTA);
GUI_Clear();
```

## 9.2 The color bar test routine

The color bar program below is used to show 13 color bars as follows:

Black -> Red, White -> Red, Black -> Green, White -> Green, Black -> Blue, White -> Blue, Black -> White, Black -> Yellow, White -> Yellow, Black -> Cyan, White -> Cyan, Black -> Magenta and White -> Magenta.

This little routine may be used on all displays in any color format. Of course, the results vary depending on the colors that can be displayed; the routine requires a display size of 320\*240 in order to show all colors. The routine is used to demonstrate the effect of the different color settings for displays. It may also be used by a test program to verify the functionality of the display, to check available colors and grayscales, as well as to correct color conversion. The screen shots are taken from the windows simulation and will look exactly like the actual output on your display if your settings and hardware are working properly. The routine is available as `COLOR_ShowColorBar.c` in the samples shipped with  $\mu$ C/GUI.

```
/* *****
*                               Micrium Inc.                               *
*      Empowering embedded systems                                     *
*                                $\mu$ C/GUI sample code                               *
* *****
*****
-----
```

```
File      : COLOR_ShowColorBar.c
Purpose   : Example draws a color bar
```

```
-----
*/

#include "GUI.H"

/*****
 *
 *           Draws 13 color bars
 *
 *****/

void ShowColorBar(void) {
    int x0 = 60, y0 = 40, yStep = 15, i;
    int NumColors = LCD_GetDevCap(LCD_DEVCAP_NUMCOLORS);
    int xsize     = LCD_GetDevCap(LCD_DEVCAP_XSIZE) - x0;
    GUI_SetFont(&GUI_Font13HB_1);
    GUI_DispStringHCenterAt("µC/GUI-sample: Show color bars", 160, 0);
    GUI_SetFont(&GUI_Font8x8);
    GUI_SetColor(GUI_WHITE);
    GUI_SetBkColor(GUI_BLACK);
    #if (LCD_FIXEDPALETTE)
        GUI_DispString("Fixed palette: ");
        GUI_DispDecMin(LCD_FIXEDPALETTE);
    #endif
    GUI_DispStringAt("Red",      0, y0 + yStep);
    GUI_DispStringAt("Green",    0, y0 + 3 * yStep);
    GUI_DispStringAt("Blue",     0, y0 + 5 * yStep);
    GUI_DispStringAt("Grey",     0, y0 + 6 * yStep);
    GUI_DispStringAt("Yellow",   0, y0 + 8 * yStep);
    GUI_DispStringAt("Cyan",     0, y0 + 10 * yStep);
    GUI_DispStringAt("Magenta",  0, y0 + 12 * yStep);
    for (i = 0; i < xsize; i++) {
        U16 cs = (255 * (U32)i) / xsize;
        U16 x = x0 + i;
        /* Red */
        GUI_SetColor(cs);
        GUI_DrawVLine(x, y0, y0 + yStep - 1);
        GUI_SetColor(0xff + (255 - cs) * 0x10100L);
        GUI_DrawVLine(x, y0 + yStep, y0 + 2 * yStep - 1);
        /* Green */
        GUI_SetColor(cs<<8);
        GUI_DrawVLine(x, y0 + 2 * yStep, y0 + 3 * yStep - 1);
        GUI_SetColor(0xff00 + (255 - cs) * 0x10001L);
        GUI_DrawVLine(x, y0 + 3 * yStep, y0 + 4 * yStep - 1);
        /* Blue */
        GUI_SetColor(cs * 0x10000L);
        GUI_DrawVLine(x, y0 + 4 * yStep, y0 + 5 * yStep - 1);
        GUI_SetColor(0xff0000 + (255 - cs) * 0x101L);
        GUI_DrawVLine(x, y0 + 5 * yStep, y0 + 6 * yStep - 1);
        /* Gray */
        GUI_SetColor((U32)cs * 0x10101L);
        GUI_DrawVLine(x, y0 + 6 * yStep, y0 + 7 * yStep - 1);
        /* Yellow */
        GUI_SetColor(cs * 0x101);
        GUI_DrawVLine(x, y0 + 7 * yStep, y0 + 8 * yStep - 1);
        GUI_SetColor(0xffff + (255 - cs) * 0x10000L);
        GUI_DrawVLine(x, y0 + 8 * yStep, y0 + 9 * yStep - 1);
        /* Cyan */
        GUI_SetColor(cs * 0x10100L);
        GUI_DrawVLine(x, y0 + 9 * yStep, y0 + 10 * yStep - 1);
        GUI_SetColor(0xffff00 + (255 - cs) * 0x1L);
        GUI_DrawVLine(x, y0 + 10 * yStep, y0 + 11 * yStep - 1);
        /* Magenta */
        GUI_SetColor(cs * 0x10001);
        GUI_DrawVLine(x, y0 + 11 * yStep, y0 + 12 * yStep - 1);
        GUI_SetColor(0xff00ff + (255 - cs) * 0x100L);
        GUI_DrawVLine(x, y0 + 12 * yStep, y0 + 13 * yStep - 1);
    }
}
```

```

/*****
 *
 *          main
 *
 *****/

void main(void) {
    GUI_Init();
    ShowColorBar();
    while(1)
        GUI_Delay(100);
}
```

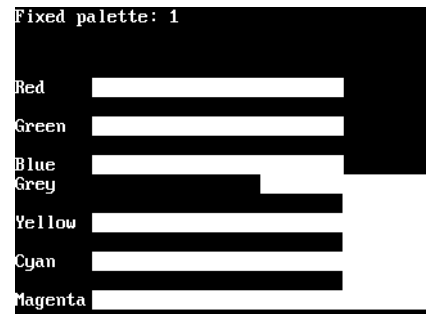
### 9.3 Fixed palette modes

The following table lists the available fixed palette color modes and the necessary #defines which need to be made in the file `LCDCConf.h` in order to select them. Detailed descriptions follow.

Color mode	No. available colors	LCD_FIXEDPALETTE	LCD_SWAP_RB
1	2 (black and white)	1	0
2	4 (grayscale)	2	0
4	16 (grayscale)	4	0
111	8	111	0
222	64	222	0
233	256	233	0
-233	256	233	1
323	256	323	0
-323	256	323	1
332	256	332	0
-332	256	332	1
444	4096	444	0
555	32768	555	0
-555	32768	555	1
565	65536	565	0
-565	65536	565	1
8666	232	8666	0

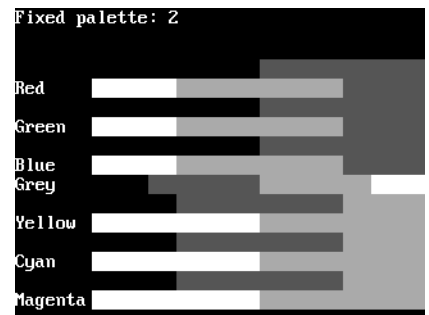
### 1 mode: 1 bpp (black and white)

Use of this mode is necessary for monochrome displays with 1 bit per pixel.  
Number of available colors: 2.



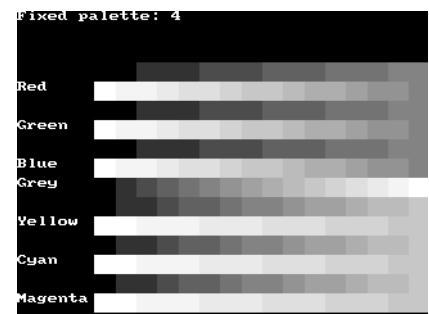
### 2 mode: 2 bpp (4 grayscales)

Use of this mode is necessary for monochrome displays with 2 bits per pixel.  
Number of available colors:  $2 \times 2 = 4$ .



### 4 mode: 4 bpp (16 grayscales)

Use of this mode is necessary for monochrome displays with 4 bits per pixel.  
Number of available colors:  $2 \times 2 \times 2 \times 2 = 16$ .



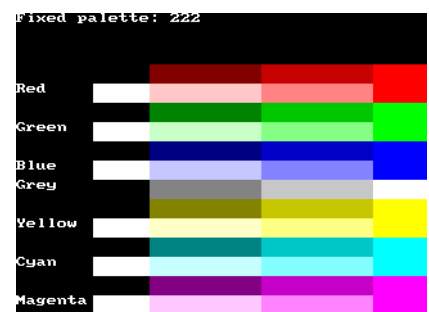
### 111 mode: 3 bpp (2 levels per color)

Use this mode if the basic 8 colors are enough, if your hardware supports only one bit per pixel and color or if you do not have sufficient video memory for a higher color depth.  
Number of available colors:  $2 \times 2 \times 2 = 8$ .



### 222 mode: 6 bpp (4 levels per color)

This mode is a good choice if your hardware does not have a palette for every individual color. 2 bits per pixel and color are reserved; usually 1 byte is used to store one pixel.  
Number of available colors:  $4 \times 4 \times 4 = 64$ .

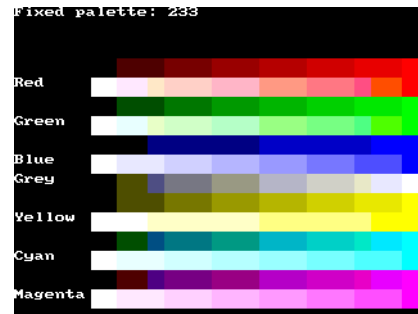


### 233 mode: 2 bits blue, 3 bits green, 3 bits red

This mode supports 256 colors. 3 bits are used for the red and green components of the color and 2 bits for the blue component. As shown in the picture, the result is 8 grades for green and red and 4 grades for blue.

Number of available colors:  $4 \times 8 \times 8 = 256$ .

Color sequence: BBGGRRR



### -233 mode: 2 bits red, 3 bits green, 3 bits blue, red and blue swapped

This mode supports 256 colors. 3 bits are used for the red and green components of the color and 2 bits for the blue component. The available colors are the same as those in 332 mode. The result is 8 grades for green and blue and 4 grades for red.

Number of available colors:  $4 \times 8 \times 8 = 256$ .

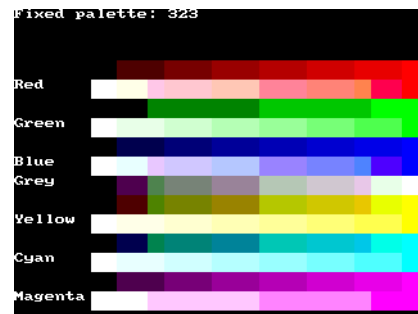
Color sequence: RRGGBBB

### 323 mode: 3 bits blue, 2 bits green, 3 bits red

This mode supports 256 colors. 3 bits are used for the red and blue components of the color and 2 bits for the green component. As shown in the picture, the result is 8 grades for blue and red and 4 grades for green.

Number of available colors:  $8 \times 4 \times 8 = 256$ .

Color sequence: BBBGGRRR



### -323 mode: 3 bits red, 2 bits green, 3 bits blue, red and blue swapped

This mode supports 256 colors. 3 bits are used for the red and blue components of the color and 2 bits for the green component. The available colors are the same as those in 323 mode. The result is 8 grades for red and blue and 4 grades for green.

Number of available colors:  $8 \times 4 \times 8 = 256$ .

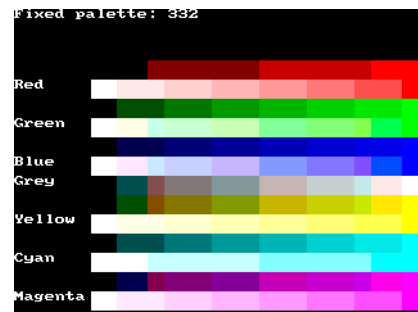
Color sequence: RRRGGBBB

### 332 mode: 3 bits blue, 3 bits green, 2 bits red

This mode supports 256 colors. 3 bits are used for the blue and green components of the color and 2 bits for the red component. As shown in the picture, the result is 8 grades for green and blue and 4 grades for red.

Number of available colors:  $8 \times 8 \times 4 = 256$ .

Color sequence: BBBGGGRR



### -332 mode: 3 bits red, 3 bits green, 2 bits blue, red and blue swapped

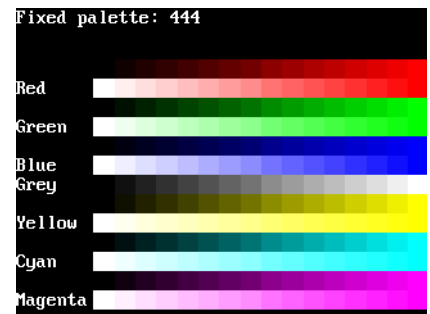
This mode supports 256 colors. 3 bits are used for the red and green components of the color and 2 bits for the blue component. The available colors are the same as those in 233 mode. The result is 8 grades for red and green and only 4 grades for blue.

Number of available colors:  $8 \times 8 \times 2 = 256$ .

Color sequence: RRRGGGBB

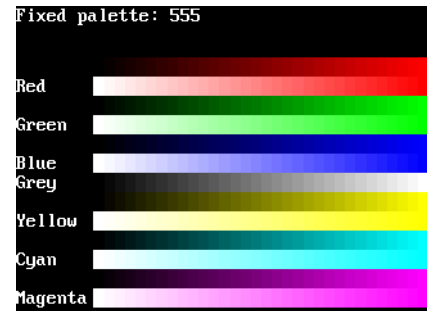
#### 444 mode: 4 bits red, 4 bits green, 4 bits blue

The red, green and blue shares are each 4 bits.  
 Number of available colors:  $16 \times 16 \times 16 = 4096$ .  
 Color sequence: BBBBGGGGRRRR



#### 555 mode: 5 bits red, 5 bits green, 5 bits blue

Use of this mode is necessary for an LCD controller that supports RGB colors with a color-depth of 15 bpp (such as SED1356 or SED13806). The red, green and blue shares are each 5 bits.  
 Number of available colors:  $32 \times 32 \times 32 = 32768$ .  
 Color sequence: BBBBGGGGRRRRR

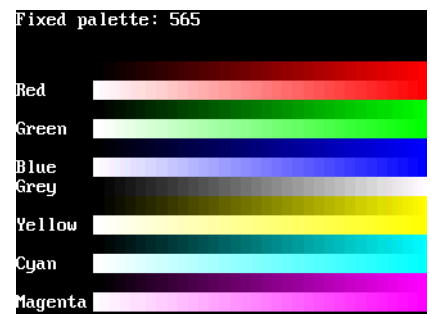


#### -555 mode: 5 bits blue, 5 bits green, 5 bits red, red and blue swapped

Use of this mode is necessary for an LCD controller that supports RGB colors with a color-depth of 15 bpp. The red, green and blue shares are each 5 bits. The available colors are the same as those in 555 mode.  
 Number of available colors:  $32 \times 32 \times 32 = 32768$ .  
 Color sequence: RRRRRGGGGGBBBBB

#### 565 mode: 5 bits red, 6 bits green, 5 bits blue

Use of this mode is necessary for an LCD controller that supports RGB colors with a color-depth of 16 bpp. The red, green and blue shares are each 5 bits. One bit remains unused.  
 Number of available colors:  $32 \times 64 \times 32 = 65536$ .  
 Color sequence: BBBBGGGGGGRRRRR



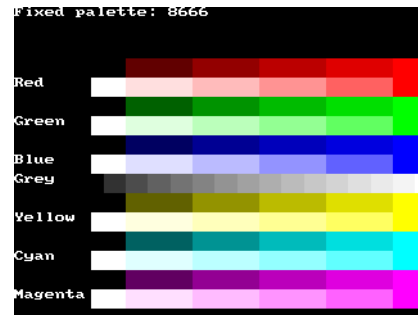
#### -565 mode: 5 bits blue, 6 bits green, 5 bits red, red and blue swapped

Use of this mode is necessary for an LCD controller that supports RGB colors with a color-depth of 16 bpp. The red, green and blue shares are each 5 bits. One bit remains unused. The available colors are the same as those in 565 mode.  
 Number of available colors:  $32 \times 64 \times 32 = 65536$ .  
 Color sequence: RRRRRGGGGGBBBBB

## 8666 mode: 8bpp, 6 levels per color + 16 gray-scales

This mode is most frequently used with a programmable color lookup table (LUT), supporting a total of 256 possible colors using a palette. The screen shot gives an idea of the available colors; this mode contains the best choice for general purpose applications. Six levels of intensity are available for each color, in addition to 16 grayscales.

Number of available colors:  $6 \times 6 \times 6 + 16 = 232$ .



## 9.4 Custom palette modes

µC/GUI can handle a custom hardware palette. A custom palette simply lists all the available colors in the same order as they are used by the hardware. This means that no matter what colors your LCD controller/display combination is able to display, µC/GUI will be able to simulate them in the PC simulation and handle these colors correctly in your target system.

In order to define a custom palette, you should do so in the configuration file `LCD-Conf.h`.

### Example

The following example (part of `LCDConf.h`) would define a custom palette with 4 colors, all of which are shades of gray:

```
#define LCD_FIXEDPALETTE 0
#define LCD_PHYS_COLORS 0xffffffff, 0xaaaaaaaa, 0x55555555, 0x00000000
```

## 9.5 Modifying the color lookup table at run time

The color information at each pixel is stored either in RGB mode (in which the red, green and blue components are kept for each pixel) or in color-index mode (in which a single number called the color index is stored for each pixel). Each color index corresponds to an entry in a lookup table, or color map, that defines a specific set of R, G and B values.

If your LCD controller features a color lookup table (LUT), it is properly initialized by µC/GUI during the initialization phase (`GUI_Init()` → `LCD_Init()` → `LCD_InitLUT()` → `LCD_L0_SetLUTEntry()`). However, it might be desirable (for various reasons) to modify the LUT at run time. Some possible reasons include:

- Color corrections in order to compensate for display problems (non-linearities) or gamma-correction
- Inversion of the display.
- The need to use more colors (at different times) than the hardware can show (at one time).

If you are simply modifying the LUT at run time, the color conversion routines will not be aware of this and will therefore still assume that the LUT is initialized as it was originally.

## Using different colors

The default contents of the color table are defined at compile time in the configuration file `GUIConf.h` (`LCD_PHYS_COLORS`). In order to minimize RAM consumption, this data is normally declared `const` and is therefore stored in ROM. In order to be able to modify it, it needs to be stored in RAM. This can be achieved by activation of the configuration switch `LCD_LUT_IN_RAM`. If this is enabled, the API function `GUI_SetLUTColor()` becomes available and can be used to modify the contents of the color table and the LUT at the same time.

A call to `LCD_InitLUT()` will restore the original (default) settings.

## 9.6 Color API

The following table lists the available color-related functions in alphabetical order within their respective categories. Detailed description of the routines can be found in the sections that follow.

Routine	Explanation
Basic color functions	
<code>GUI_GetBkColor()</code>	Return the current background color.
<code>GUI_GetBkColorIndex()</code>	Return the index of the current background color.
<code>GUI_GetColor()</code>	Return the current foreground color.
<code>GUI_GetColorIndex()</code>	Return the index of the current foreground color.
<code>GUI_SetBkColor()</code>	Set the current background color.
<code>GUI_SetBkColorIndex()</code>	Set the index of the current background color.
<code>GUI_SetColor()</code>	Set the current foreground color.
<code>GUI_SetColorIndex()</code>	Set the index of the current foreground color.
Index & color conversion	
<code>GUI_Color2Index()</code>	Convert color into color index.
<code>GUI_Index2Color()</code>	Convert color index into color.
Lookup table (LUT) group	
<code>GUI_InitLUT()</code>	Initialize the LUT (hardware).
<code>GUI_SetLUTColor()</code>	Set color of a color index (both hardware and color table).
<code>GUI_SetLUTEntry()</code>	Write a value into the LUT (hardware).

## 9.7 Basic color functions

### GUI\_GetBkColor()

#### Description

Returns the current background color.

#### Prototype

```
GUI_COLOR GUI_GetBkColor(void);
```

#### Return value

The current background color.

## GUI\_GetBkColorIndex()

**Description**  
Returns the index of the current background color.

**Prototype**  
`int GUI_GetBkColorIndex(void);`

**Return value**  
The current background color index.

## GUI\_GetColor()

**Description**  
Returns the current foreground color.

**Prototype**  
`GUI_COLOR GUI_GetColor(void);`

**Return value**  
The current foreground color.

## GUI\_GetColorIndex()

**Description**  
Returns the index of the current foreground color.

**Prototype**  
`int GUI_GetColorIndex(void);`

**Return value**  
The current foreground color index.

## GUI\_SetBkColor()

**Description**  
Sets the current background color.

**Prototype**  
`GUI_COLOR GUI_SetBkColor(GUI_COLOR Color);`

Parameter	Meaning
Color	Color for background, 24-bit RGB value.

**Return value**  
The selected background color.

## GUI\_SetBkColorIndex()

**Description**  
Sets the index of the current background color.

**Prototype**

```
int GUI_SetBkColorIndex(int Index);
```

Parameter	Meaning
<a href="#">Index</a>	Index of the color to be used.

**Return value**

The selected background color index.

**GUI\_SetColor()****Description**

Sets the current foreground color.

**Prototype**

```
void GUI_SetColor(GUI_COLOR Color);
```

Parameter	Meaning
<a href="#">Color</a>	Color for foreground, 24-bit RGB value.

**Return value**

The selected foreground color.

**GUI\_SetColorIndex()****Description**

Sets the index of the current foreground color.

**Prototype**

```
void GUI_SetColorIndex(int Index);
```

Parameter	Meaning
<a href="#">Index</a>	Index of the color to be used.

**Return value**

The selected foreground color index.

**9.8 Index & color conversion****GUI\_Color2Index()**

Returns the index of a specified RGB color value.

**Prototype**

```
int GUI_Color2Index(GUI_COLOR Color)
```

Parameter	Meaning
<a href="#">Color</a>	RGB value of the color to be converted.

**Return value**

The color index.

**GUI\_Index2Color()**

Returns the RGB color value of a specified index.

**Prototype**

```
int GUI_Index2Color(int Index)
```

Parameter	Meaning
Index	Index of the color. to be converted

**Return value**

The RGB color value.

**9.9 Lookup table (LUT) group**

These functions are optional and will work only if supported by the LCD controller hardware. An LCD controller with LUT hardware is required. Please consult the manual for the LCD controller you are using for more information on LUTs.

**GUI\_InitLUT()**

**Description**

Initializes the lookup table of the LCD controller(s).

**Prototype**

```
void LCD_InitLUT(void);
```

**Additional information**

The lookup table needs to be enabled (by the LCD\_INITCONTROLLER macro) for this function to have any effect.

**GUI\_SetLUTColor()**

**Description**

Modifies a single entry to the color table and the LUT of the LCD controller(s).

**Prototype**

```
void GUI_SetLUTColor(U8 Pos, GUI_COLOR Color);
```

Parameter	Meaning
Pos	Position within the lookup table. Should be less than the number of colors (e.g. 0-3 for 2 bpp, 0-15 for 4 bpp, 0-255 for 8 bpp).
Color	24-bit RGB value.

## Additional information

The closest value possible will be used for the LUT. If a color LUT is to be initialized, all 3 components are used. In monochrome modes the green component is used, but it is still recommended (for better understanding of the program code) to set all 3 colors to the same value (such as 0x555555 or 0xa0a0a0).

The lookup table needs to be enabled (by the `LCD_INITCONTROLLER` macro) for this function to have any effect. This function is always available, but has an effect only if:

- a) The LUT is used
- b) The color table is located in RAM (`LCD_PHYSCOLORS_IN_RAM`)

## GUI\_SetLUTEntry()

### Description

Modifies a single entry to the LUT of the LCD controller(s).

### Prototype

```
void GUI_SetLUTEntry(U8 Pos, GUI_COLOR Color);
```

Parameter	Meaning
Pos	Position within the lookup table. Should be less than the number of colors (e.g. 0-3 for 2 bpp, 0-15 for 4 bpp, 0-255 for 8 bpp).
Color	24-bit RGB value.

## Additional information

The closest value possible will be used for the LUT. If a color LUT is to be initialized, all 3 components are used. In monochrome modes the green component is used, but it is still recommended (for better understanding of the program code) to set all 3 colors to the same value (such as 0x555555 or 0xa0a0a0).

The lookup table needs to be enabled (by the `LCD_INITCONTROLLER` macro) for this function to have any effect. This function is often used to ensure that the colors actually displayed match the logical colors (linearization).

### Example

```
//
// Linearize the palette of a 4-grayscale LCD
//
GUI_SetLUTEntry(0, 0x000000);
GUI_SetLUTEntry(1, 0x777777); // 555555 would be linear
GUI_SetLUTEntry(2, 0xbbbbbb); // aaaaaa would be linear
GUI_SetLUTEntry(3, 0xffffffff);
```



# Chapter 10

## Memory Devices

---

Memory devices can be used in a variety of situations, mainly to prevent the display from flickering when using drawing operations for overlapping items. The basic idea is quite simple. Without the use of a memory device, drawing operations write directly to the display. The screen is updated as drawing operations are executed, which gives it a flickering appearance as the various updates are made. For example, if you wanted to draw a bitmap in the background and some transparent text in the foreground, you would first have to draw the bitmap and then the text. The effect would be a flickering of the text.


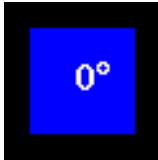
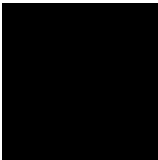

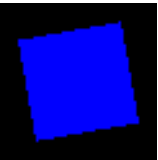

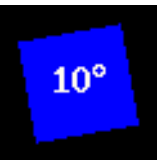
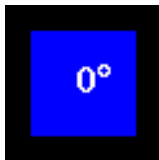

If a memory device is used for such a procedure, however, all drawing operations are executed in memory. The final result is displayed on the screen only when all operations have been carried out, with the advantage of no flickering. This difference can be seen in the example in the following section, which illustrates a sequence of drawing operations both with and without the use of a memory device.

The distinction may be summarized as follows: If no memory device is used, the effects of drawing operations can be seen step by step, with the disadvantage of a flickering display. With a memory device, the effects of all routines are made visible as a single operation. No intermediate steps can actually be seen. The advantage, as explained above, is that display flickering is completely eliminated, and this is often desirable.

Memory devices are an additional (optional) software item and are not shipped with the  $\mu$ C/GUI basic package. The software for memory devices is located in the subdirectory `GUI\Memdev`.

# 10.1 Using memory devices: an illustration

The following table shows screen shots of the same operations handled with and without a memory device. The objective in both cases is identical: a workpiece is to be rotated and labeled with the respective angle of rotation (here, 10 degrees). In the first case (without a memory device) the screen must be cleared, then the polygon is redrawn in the new position and a string with the new label is written. In the second case (with a memory device) the same operations are performed in memory, but the screen is not updated during this time. The only update occurs when the routine `GUI_MEMDEV_CopyToLCD` is called, and this update reflects all the operations at once. Note that the initial states and final outputs of both procedures are identical.

API function	Without memory device	With memory device
Step 0: Initial state		
Step 1: GUI_Clear		
Step 2: GUI_DrawPolygon		
Step 3: GUI_DispString		
Step 4: GUI_MEMDEV_CopyToLCD (only when using memory device)		

## 10.2 Basic functions

The following routines are those that are normally called when using memory devices. Basic usage is rather simple:

1. Create the memory device (using `GUI_MEMDEV_Create()`).
2. Activate it (using `GUI_MEMDEV_Select()`).
3. Execute drawing operations.
4. Copy the result into the display (using `GUI_MEMDEV_CopyToLCD()`).
5. Delete the memory device if you no longer need it (using `GUI_MEMDEV_Delete()`).

## 10.3 In order to be able to use memory devices...

Memory devices are enabled by default. In order to optimize performance of the software, support for memory devices can be switched off in the configuration file `GUIConf.h` by including the following line:

```
#define GUI_SUPPORT_MEMDEV 0
```

If this line is in the configuration file and you want to use memory devices, either delete the line or change the define to 1.

## 10.4 Memory device API

The table below lists the available routines of the  $\mu$ C/GUI memory device API. All functions are listed in alphabetical order within their respective categories. Detailed descriptions of the routines can be found in the sections that follow.

Routine	Explanation
Basic functions	
<code>GUI_MEMDEV_Create()</code>	Create the memory device (first step).
<code>GUI_MEMDEV_CopyToLCD()</code>	Copy contents of memory device to LCD.
<code>GUI_MEMDEV_Delete()</code>	Free the memory used by the memory device.
<code>GUI_MEMDEV_Select()</code>	Select a memory device as target for drawing operations.
Advanced features	
<code>GUI_MEMDEV_Clear()</code>	Mark the memory device contents as unchanged.
<code>GUI_MEMDEV_CopyFromLCD()</code>	Copy contents of LCD to memory device.
<code>GUI_MEMDEV_CopyToLCDAA()</code>	Copy the contents of memory device antialiased.
<code>GUI_MEMDEV_GetYSize()</code>	Return the Y-size of memory device.
<code>GUI_MEMDEV_ReduceYSize()</code>	Reduce Y-size of memory device.
<code>GUI_MEMDEV_SetOrg()</code>	Change the origin of the memory device on the LCD.
Banding memory device	
<code>GUI_MEMDEV_Draw()</code>	Use a memory device for drawing.
Auto device object functions	

Routine	Explanation
GUI_MEMDEV_CreateAuto( )	Create an auto device object.
GUI_MEMDEV_DeleteAuto( )	Delete an auto device object.
GUI_MEMDEV_DrawAuto( )	Use a GUI_AUTODEV object for drawing.

## GUI\_MEMDEV\_Create()

### Description

Creates a memory device.

### Prototype

```
GUI_MEMDEV_Handle GUI_MEMDEV_Create(int x0, int y0, int XSize, int YSize)
```

Parameter	Meaning
x0	X-position of memory device.
y0	Y-position of memory device.
xsize	X-size of memory device.
ysize	Y-size of memory device.

### Return value

Handle for created memory device. If the routine fails the return value is 0.

## GUI\_MEMDEV\_CopyToLCD()

### Description

Copies the contents of a memory device from memory to the LCD.

### Prototype

```
void GUI_MEMDEV_CopyToLCD(GUI_MEMDEV_Handle hMem)
```

Parameter	Meaning
hMem	Handle to memory device.

## GUI\_MEMDEV\_Delete()

### Description

Deletes a memory device.

### Prototype

```
void GUI_MEMDEV_Delete(GUI_MEMDEV_Handle MemDev);
```

Parameter	Meaning
hMem	Handle to memory device.

### Return value

Handle for deleted memory device.

## GUI\_MEMDEV\_Select()

### Description

Activates a memory device (or activates LCD if handle is 0)

## Prototype

```
void GUI_MEMDEV_Select(GUI_MEMDEV_Handle hMem)
```

Parameter	Meaning
<a href="#">hMem</a>	Handle to memory device.

## 10.5 Advanced features

### GUI\_MEMDEV\_Clear()

#### Description

Marks the entire contents of a memory device as "unchanged".

#### Prototype

```
void GUI_MEMDEV_Clear(GUI_MEMDEV_Handle hMem);
```

Parameter	Meaning
<a href="#">hMem</a>	Handle to memory device.

#### Additional Information

The next drawing operation with `GUI_MEMDEV_CopyToLCD` will then write only the bytes modified between `GUI_MEMDEV_Clear` and `GUI_MEMDEV_CopyToLCD`.

### GUI\_MEMDEV\_CopyFromLCD()

#### Description

Copies the contents of a memory device from LCD data (video memory) to the memory device. In other words: read back the contents of the LCD to the memory device.

#### Prototype

```
void GUI_MEMDEV_CopyFromLCD(GUI_MEMDEV_Handle hMem);
```

Parameter	Meaning
<a href="#">hMem</a>	Handle to memory device.

### GUI\_MEMDEV\_CopyToLCDAA()

#### Description

Copies the contents of a memory device (antialiased) from memory to the LCD.

#### Prototype

```
void GUI_MEMDEV_CopyToLCDAA(GUI_MEMDEV_Handle MemDev);
```

Parameter	Meaning
<a href="#">hMem</a>	Handle to memory device.

## Additional Information

The device data is handled as antialiased data. A matrix of 2x2 pixels is converted to 1 pixel. The intensity of the resulting pixel depends on how many pixels are set in the matrix.

## Example

Creates a memory device and selects it for output. A large font is then set and a text is written to the memory device:

```
GUI_MEMDEV_Handle hMem = GUI_MEMDEV_Create(0,0,60,32);
GUI_MEMDEV_Select(hMem);
GUI_SetFont(&GUI_Font32B_ASCII);
GUI_DispString("Text");
GUI_MEMDEV_CopyToLCDAA(hMem);
```

## Screen shot for preceeding example



## GUI\_MEMDEV\_GetYSize()

### Description

Returns the Y-size of a memory device.

### Prototype

```
int GUI_MEMDEV_GetYSize(GUI_MEMDEV_Handle hMem);
```

Parameter	Meaning
<a href="#">hMem</a>	Handle to memory device.

## GUI\_MEMDEV\_ReduceYSize()

### Description

Reduces the Y-size of a memory device.

### Prototype

```
void GUI_MEMDEV_ReduceYSize(GUI_MEMDEV_Handle hMem, int YSize);
```

Parameter	Meaning
<a href="#">hMem</a>	Handle to memory device.
<a href="#">YSize</a>	New Y-size of the memory device.

## Additional Information

Changing the size of the memory device is more efficient than deleting and then recreating it.

## GUI\_MEMDEV\_SetOrg

### Description

Changes the origin of the memory device on the LCD.

Prototype

```
void GUI_MEMDEV_SetOrg(GUI_MEMDEV_Handle hMem, int x0, int y0);
```

Parameter	Meaning
<a href="#">hMem</a>	Handle to memory device.
<a href="#">x0</a>	Horizontal position (of the upper left pixel).
<a href="#">y0</a>	Vertical position (of the upper left pixel).

Additional Information

This routine can be helpful when the same device is used for different areas of the screen or when the contents of the memory device are to be copied into different areas.

Changing the origin of the memory device is more efficient than deleting and then recreating it.

## Simple example for using a memory device

This example demonstrates the use of a basic memory device. Text is written into a memory device and then copied to the LCD. It can be found under Source\Misc\MemDev.c.

```

/*****
*
*                      Micrium Inc.
*                      Empowering embedded systems
*
*                      µC/GUI sample code
*
*****/

-----
File      : MemDev.c
Purpose   : Simple demo shows the use of memory devices
-----
*/

#include "GUI.H"

/*****
*
*                      Shows the use of memory devices
*
*****/

static void DemoMemDev(void) {
    GUI_MEMDEV_Handle hMem;
    while(1) {
        /* Create the memory device... */
        hMem = GUI_MEMDEV_Create(0, 0, 110, 18);
        /* ...and select it for drawing operations */
        GUI_MEMDEV_Select(hMem);
        /* Draw text to memory device */
        GUI_SetFont(&GUI_FontComic18B_ASCII);
        GUI_DispStringAt("Memory device", 0, 0);
        /* Copy memory device contents to LCD */
        GUI_MEMDEV_CopyToLCD(hMem);
        /* Select LCD and destroy memory device */
        GUI_MEMDEV_Select(0);
        GUI_MEMDEV_Delete(hMem);
        GUI_Delay(1000);
        GUI_Clear();
        GUI_Delay(500);
    }
}

/*****
*
*                      main
*
*****/

void main(void) {
    GUI_Init();
    DemoMemDev();
}

```

## 10.6 Banding memory device

A memory device is first filled by executing the specified drawing functions. After filling the device, the contents are drawn to the LCD. There may not be enough memory available to store the complete output area at once, depending on your configuration (see the `GUI_ALLOC_SIZE` configuration macro in Chapter 21: "High-Level Configuration"). A banding memory device divides the drawing area into bands, in which each band covers as many lines as possible with the currently available memory.

### GUI\_MEMDEV\_Draw()

#### Description

Basic drawing function that prevents flickering of the display.

#### Prototype

```
int GUI_MEMDEV_Draw(GUI_RECT* pRect,
                    GUI_CALLBACK_VOID_P* pfDraw,
                    void* pData,
                    int NumLines,
                    int Flags)
```

Parameter	Meaning
<code>pRect</code>	Pointer to a <code>GUI_RECT</code> structure for the used LCD area.
<code>pfDraw</code>	Pointer to a callback function for executing the drawing.
<code>pData</code>	Pointer to a data structure used as parameter for the callback function.
<code>NumLines</code>	0 (recommended) or number of lines for the memory device.
<code>Flags</code>	0 or <code>GUI_MEMDEV_HASTRANS</code> .

#### Return value

0 if successful, 1 if the routine fails.

#### Additional Information

If the parameter `NumLines` is 0, the number of lines in each band is calculated automatically by the function. The function then iterates over the output area band by band by moving the origin of the memory device.

### Example for using a banding memory device

The following example demonstrates the use of a banding memory device. It can be found under `Source\Misc\BandingMemdev.c`.

```

/*****
*                               Micrium Inc.                               *
*                               Empowering embedded systems               *
*                               *                                           *
*                               µC/GUI sample code                       *
*                               *                                           *
*****/

-----
File       : BandingMemdev.c
Purpose    : Example demonstrating the use of banding memory devices
-----
*/

#include "gui.h"
```

```

static const GUI_POINT aPoints[] = {
    {-50, 0},
    {-10, 10},
    {0, 50},
    {10, 10},
    {50, 0},
    {10, -10},
    {0, -50},
    {-10, -10}
};

#define SIZE_OF_ARRAY(Array) (sizeof(Array) / sizeof(Array[0]))

typedef struct {
    int XPos_Poly, YPos_Poly;
    int XPos_Text, YPos_Text;
    GUI_POINT aPointsDest[8];
} tDrawItContext;

/*****
 *
 *           Drawing routine
 *
 *****/

static void DrawIt(void * pData) {
    tDrawItContext * pDrawItContext = (tDrawItContext *)pData;
    GUI_Clear();
    GUI_SetFont(&GUI_Font8x8);
    GUI_SetTextMode(GUI_TM_TRANS);
    /* draw background */
    GUI_SetColor(GUI_GREEN);
    GUI_FillRect(pDrawItContext->XPos_Text,
                pDrawItContext->YPos_Text - 25,
                pDrawItContext->XPos_Text + 100,
                pDrawItContext->YPos_Text - 5);

    /* draw polygon */
    GUI_SetColor(GUI_BLUE);
    GUI_FillPolygon(pDrawItContext->aPointsDest, SIZE_OF_ARRAY(aPoints), 160, 120);
    /* draw foreground */
    GUI_SetColor(GUI_RED);
    GUI_FillRect(220 - pDrawItContext->XPos_Text,
                pDrawItContext->YPos_Text + 5,
                220 - pDrawItContext->XPos_Text + 100,
                pDrawItContext->YPos_Text + 25);
}

/*****
 *
 *           Demonstrates the banding memory device
 *
 *****/

#define USE_BANDING_MEMDEV (1) /* Set to 0 for drawing without banding memory device */

void DemoBandingMemdev(void) {
    int i;
    int XSize = LCD_GET_XSIZE();
    int YSize = LCD_GET_YSIZE();
    tDrawItContext DrawItContext;
    GUI_SetFont(&GUI_Font8x9);
    GUI_SetColor(GUI_WHITE);
    GUI_DispStringHCenterAt("Banding memory device\nwithout flickering",
                            XSize / 2, 40);
    DrawItContext.XPos_Poly = XSize / 2;
    DrawItContext.YPos_Poly = YSize / 2;
    DrawItContext.YPos_Text = YSize / 2 - 4;
    for (i = 0; i < (XSize - 100); i++) {
        float angle = i * 3.1415926 / 60;
        DrawItContext.XPos_Text = i;
    }
}

```

```

/* Rotate the polygon */
GUI_RotatePolygon(DrawItContext.aPointsDest,
                  aPoints,
                  SIZE_OF_ARRAY(aPoints), angle);
#if USE_BANDING_MEMDEV
{
    GUI_RECT Rect = {0, 70, 320,170};
    /* Use banding memory device for drawing */
    GUI_MEMDEV_Draw(&Rect, &DrawIt, &DrawItContext, 0, 0);
}
#else
/* Simple drawing without using memory devices */
DrawIt((void *)&DrawItContext);
#endif
#ifdef WIN32
    GUI_Delay(20); /* Use a short delay only in the simulation */
#endif
}
}

/*****
*
*           main
*
*****/

void main(void) {
    GUI_Init();
    while(1) {
        DemoBandingMemdev();
    }
}

```

### Screen shot for preceeding example



## 10.7 Auto device object

Memory devices are useful when the display must be updated to reflect the movement or changing of items, since it is important in such applications to prevent the LCD from flickering. An auto device object is based on the banding memory device, and may be more efficient for applications such as moving indicators, in which only a small part of the display is updated at a time.

The device automatically distinguishes which areas of the display consist of fixed objects and which areas consist of moving or changing objects that must be updated. When the drawing function is called for the first time, all items are drawn. Each fur-

ther call updates only the space used by the moving or changing objects. The actual drawing operation uses the banding memory device, but only within the necessary space. The main advantage of using an auto device object (versus direct usage of a banding memory device) is that it saves computation time, since it does not keep updating the entire display.

## GUI\_MEMDEV\_CreateAuto()

### Description

Creates an auto device object.

### Prototype

```
int GUI_MEMDEV_CreateAuto(GUI_AUTODEV * pAutoDev);
```

Parameter	Meaning
<a href="#">pAutoDev</a>	Pointer to a GUI_AUTODEV object.

### Return value

Currently 0, reserved for later use.

## GUI\_MEMDEV\_DeleteAuto()

### Description

Deletes an auto device object.

### Prototype

```
void GUI_MEMDEV_DeleteAuto(GUI_AUTODEV * pAutoDev);
```

Parameter	Meaning
<a href="#">pAutoDev</a>	Pointer to a GUI_AUTODEV object.

## GUI\_MEMDEV\_DrawAuto()

### Description

Executes a specified drawing routine using a banding memory device.

### Prototype

```
int GUI_MEMDEV_DrawAuto(GUI_AUTODEV * pAutoDev,
                        GUI_AUTODEV_INFO * pAutoDevInfo,
                        GUI_CALLBACK_VOID_P * pfDraw,
                        void * pData);
```

Parameter	Meaning
<a href="#">pAutoDev</a>	Pointer to a GUI_AUTODEV object.
<a href="#">pAutoDevInfo</a>	Pointer to a GUI_AUTODEV_INFO object.
<a href="#">pfDraw</a>	Pointer to the user-defined drawing function which is to be executed.
<a href="#">pData</a>	Pointer to a data structure passed to the drawing function.

### Return value

0 if successful, 1 if the routine fails.

## Additional Information

The GUI\_AUTODEV\_INFO structure contains the information about what items must be drawn by the user function:

```
typedef struct {
    char DrawFixed;
} GUI_AUTODEV_INFO;
```

DrawFixed is set to 1 if all items have to be drawn. It is set to 0 when only the moving or changing objects have to be drawn. We recommend the following procedure when using this feature:

```
typedef struct {
    GUI_AUTODEV_INFO AutoDevInfo; /* Information about what has to be drawn */
    /* Additional data used by the user function */
    ...
} PARAM;

static void Draw(void * p) {
    PARAM * pParam = (PARAM *)p;
    if (pParam->AutoDevInfo.DrawFixed) {
        /* Draw fixed background */
        ...
    }
    /* Draw moving objects */
    ...
    if (pParam->AutoDevInfo.DrawFixed) {
        /* Draw fixed foreground (if needed) */
        ...
    }
}

void main(void) {
    PARAM Param; /* Parameters for drawing routine */
    GUI_AUTODEV AutoDev; /* Object for banding memory device */
    /* Set/modify informations for drawing routine */
    ...
    GUI_MEMDEV_CreateAuto(&AutoDev); /* Create GUI_AUTODEV-object */
    GUI_MEMDEV_DrawAuto(&AutoDev, /* Use GUI_AUTODEV-object for drawing */
        &Param.AutoDevInfo,
        &Draw,
        &Param);
    GUI_MEMDEV_DeleteAuto(&AutoDev); /* Delete GUI_AUTODEV-object */
}
```

## Example for using an auto device object

The following example demonstrates the use of an auto device object. It can be found under Source\Misc\AutoDev.c. A scale with a moving needle is drawn in the background and a small text is written in the foreground. The needle is drawn with the antialiasing feature of µC/GUI. High-resolution antialiasing is used here to improve the appearance of the moving needle. For more information please see Chapter 15: "Antialiasing".

```
/* *****
 *                               Micrium Inc.                               *
 *                               Empowering embedded systems                 *
 *                               *                                           *
 *                               µC/GUI sample code                         *
 *                               *                                           *
 * *****
```

```
-----
File      : AutoDev.c
Purpose   : Example demonstrating the use of GUI_AUTODEV-objects
-----
```

```

*/

#include "gui.h"
#include <math.h>
#include <stddef.h>
#ifndef WIN32
    #include "rtos.h"
#endif

#define countof(Obj) (sizeof(Obj)/sizeof(Obj[0]))
#define DEG2RAD      (3.1415926f/180)

/*****
 *
 *                               Scale bitmap
 *
 *****/

static const GUI_COLOR ColorsScaleR140[] = {
    0x000000,0x00AA00,0xFFFFFf,0x0000AA,
    0x00FF00,0xAEAEAE,0x737373,0xD3D3D3,
    0xDFDFDF,0xBBDFBB,0x6161DF,0x61DF61,
    0BBBBBDF,0xC7C7C7,0x616193
};

static const GUI_LOGPALETTE PalScaleR140 = {
    15, /* number of entries */
    0, /* No transparency */
    &ColorsScaleR140[0]
};

static const unsigned char acScaleR140[] = {
    /* ... The pixel data is not shown in the manual.
       Please take a look at the sample source file ... */
};

static const GUI_BITMAP bmScaleR140 = {
    200,          /* XSize */
    73,           /* YSize */
    100,          /* BytesPerLine */
    4,            /* BitsPerPixel */
    acScaleR140,  /* Pointer to picture data (indices) */
    &PalScaleR140 /* Pointer to palette */
};

/*****
 *
 *                               Shape of polygon
 *
 *****/

#define MAG 3

static const GUI_POINT aNeedle[] = {
    { MAG * ( 0), MAG * ( 0 + 125) },
    { MAG * (-3), MAG * (-15 + 125) },
    { MAG * (-3), MAG * (-65 + 125) },
    { MAG * ( 3), MAG * (-65 + 125) },
    { MAG * ( 3), MAG * (-15 + 125) },
};

/*****
 *
 *                               Structure containing information for drawing routine
 *
 *****/

typedef struct {
    /* Information about what has to be displayed */
    GUI_AUTODEV_INFO AutoDevInfo;

```

```

/* Polygon data */
GUI_POINT aPoints[7];
float Angle;
} PARAM;

/*****
*
*                               GetAngle
*
*****/

This routine returns the value value to indicate. In a real application,
this value would somehow be measured.
*/

static float GetAngle(int tDiff) {
    if (tDiff < 15000) {
        return 225 - 0.006 * tDiff ;
    }
    tDiff -= 15000;
    if (tDiff < 7500) {
        return 225 - 90 + 0.012 * tDiff ;
    }
    tDiff -= 7000;
    return 225;
}

/*****
*
*                               Drawing routine
*
*****/

static void Draw(void * p) {
    PARAM * pParam = (PARAM *)p;
    /* Fixed background */
    if (pParam->AutoDevInfo.DrawFixed) {
        GUI_ClearRect (60, 50 + bmScaleR140.YSize, 60 + bmScaleR140.XSize - 1, 150);
        GUI_DrawBitmap(&bmScaleR140, 60, 50);
    }
    /* Moving needle */
    GUI_SetColor(GUI_WHITE);
    GUI_AA_FillPolygon(pParam->aPoints, countof(aNeedle), MAG * 160, MAG * 190);
    /* Fixed foreground */
    if (pParam->AutoDevInfo.DrawFixed) {
        GUI_SetTextMode(GUI_TM_TRANS);
        GUI_SetColor(GUI_RED);
        GUI_SetFont(&GUI_Font24B_ASCII);
        GUI_DispStringHCenterAt("RPM / 1000", 160, 110);
    }
}

/*****
*
*   Shows a scale with a needle using a banding memory device
*
*****/

static void DemoScale(void) {
    int Cnt;
    int tDiff, t0 = GUI_GetTime();
    PARAM Param;           /* Parameters for drawing routine */
    GUI_AUTODEV AutoDev;    /* Object for banding memory device */
    /* Show message */
    GUI_SetColor(GUI_WHITE);
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringHCenterAt("Scale using GUI_AUTODEV-object", 160, 0);
    /* Enable high resolution for antialiasing */
    GUI_AA_EnableHiRes();
    GUI_AA_SetFactor(MAG);
    /* Create GUI_AUTODEV-object */

```

```

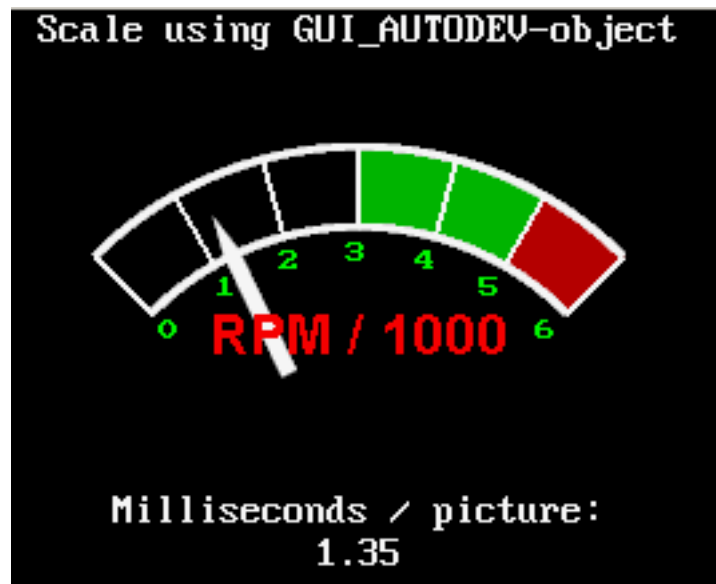
GUI_MEMDEV_CreateAuto(&AutoDev);
/* Show needle for a fixed time */
for (Cnt = 0; (tDiff = GUI_GetTime() - t0) < 24000; Cnt++) {
    /* Get value to display and calculate polygon for needle */
    Param.Angle = GetAngle(tDiff) * DEG2RAD;
    GUI_RotatePolygon(Param.aPoints, aNeedle, countof(aNeedle), Param.Angle);
    GUI_MEMDEV_DrawAuto(&AutoDev, &Param.AutoDevInfo, &Draw, &Param);
}
/* Display milliseconds / picture */
GUI_SetColor(GUI_WHITE);
GUI_SetFont(&GUI_Font8x16);
GUI_DispStringHCenterAt("Milliseconds / picture:", 160, 180);
GUI_SetTextAlign(GUI_TA_CENTER);
GUI_SetTextMode(GUI_TM_NORMAL);
GUI_DispNextLine();
GUI_GotoX(160);
GUI_DispFloatMin((float)tDiff / (float)Cnt, 2);
/* Delete GUI_AUTODEV-object */
GUI_MEMDEV_DeleteAuto(&AutoDev);
}

/*****
*
*                               main
*
*****/

void main(void) {
    #ifndef WIN32
        OS_InitKern();
        OS_InitHW();
    #endif
    GUI_Init();
    while(1)
        DemoScale();
}

```

### Screen shot for preceeding example





# Chapter 11

## Execution Model: Single Task / Multitask

---

$\mu$ C/GUI has been designed from the beginning to be compatible with different types of environments. It works in single task and in multitask applications, with a proprietary operating system or with any commercial RTOS such as embOS or uC/OS.

## 11.1 Supported execution models

We have to basically distinguish between 3 different execution models:

### Single task system (superloop)

The entire program runs in one superloop. Normally, all software components are periodically called. Interrupts must be used for real time parts of the software since no real time kernel is used.

### μC/GUI Multitask system: one task calling μC/GUI

A real time kernel (RTOS) is used, but only one task calls μC/GUI functions. From the graphic software's point of view, it is the same as being used in a single task system.

### μC/GUI Multitask system: multiple tasks calling μC/GUI

A real time kernel (RTOS) is used, and multiple tasks call μC/GUI functions. This works without a problem as long as the software is made thread-safe, which is done by enabling multitask support in the configuration and adapting the kernel interface routines. For popular kernels, the kernel interface routines are readily available.

## 11.2 Single task system (superloop)

### Description

The entire program runs in one superloop. Normally, all components of the software are periodically called. No real time kernel is used, so interrupts must be used for real time parts of the software. This type of system is primarily used in smaller systems or if real time behavior is not critical.

### Superloop example (without μC/GUI)

```
void main (void) {
    HARDWARE_Init();

    /* Init software components */
    XXX_Init();
    YYY_Init();

    /* Superloop: call all software components regularly */
    while (1) {
        /* Exec all components of the software */
        XXX_Exec();
        YYY_Exec();
    }
}
```

### Advantages

No real time kernel is used (-> smaller ROM size, just one stack -> less RAM for stacks), no preemption/synchronization problems.

### Disadvantages

The superloop type of program can become hard to maintain if it exceeds a certain program size. Real time behavior is poor, since one software component cannot be interrupted by any other component (only by interrupts). This means that the reaction time of one software component depends on the execution time of all other components in the system.

## Using $\mu$ C/GUI

There are no real restrictions regarding the use of  $\mu$ C/GUI. As always, `GUI_Init()` has to be called before you can use the software. From there on, any API function can be used. If the window manager's callback mechanism is used, then an  $\mu$ C/GUI update function has to be called regularly. This is typically done by calling the `GUI_Exec()` from within the superloop. Blocking functions such as `GUI_Delay()` and `GUI_ExecDialog()` should not be used in the loop since they would block the other software modules.

The default configuration, which does not support multitasking (`#define GUI_MT 0`) can be used; kernel interface routines are not required.

### Superloop example (with $\mu$ C/GUI)

```
void main (void) {
    HARDWARE_Init();

    /* Init software components */
    XXX_Init();
    YYY_Init();
    GUI_Init();          /* Init  $\mu$ C/GUI */

    /* Superloop: call all software components regularly */
    while (1) {
        /* Exec all components of the software */
        XXX_Exec();
        YYY_Exec();
        GUI_Exec();      /* Exec  $\mu$ C/GUI for functionality like updating windows */
    }
}
```

## 11.3 $\mu$ C/GUIMultitask system: one task calling $\mu$ C/GUI

### Description

A real time kernel (RTOS) is used. The user program is split into different parts, which execute in different tasks and typically have different priorities. Normally the real time critical tasks (which require a certain reaction time) will have the highest priorities. **One single task** is used for the user interface, which calls  $\mu$ C/GUI functions. This task usually has the lowest priority in the system or at least one of the lowest (some statistical tasks or simple idle processing may have even lower priorities).

Interrupts can, but do not have to be used for real time parts of the software.

### Advantages

The real time behavior of the system is excellent. The real time behavior of a task is affected only by tasks running at higher priority. This means that changes to a program component running in a low priority task do not affect the real time behavior at all. If the user interface is executed from a low priority task, this means that changes to the user interface do not affect the real time behavior. This kind of system makes it easy to assign different components of the software to different members of the development team, which can work to a high degree independently from each other.

## Disadvantages

You need to have a real time kernel (RTOS), which costs money and uses up ROM and RAM (for stacks). In addition, you will have to think about task synchronization and how to transfer information from one task to another.

## Using $\mu$ C/GUI

If the window manager's callback mechanism is used, then an  $\mu$ C/GUI update function (typically `GUI_Exec()`, `GUI_Delay()`) has to be called regularly from the task calling  $\mu$ C/GUI. Since  $\mu$ C/GUI is only called by one task, to  $\mu$ C/GUI it is the same as being used in a single task system.

The default configuration, which does not support multitasking (`#define GUI_MT 0`) can be used; kernel interface routines are not required. You can use any real time kernel, commercial or proprietary. $\mu$ C/GUI

# 11.4 $\mu$ C/GUI Multitask system: multiple tasks calling $\mu$ C/GUI

## Description

A real time kernel (RTOS) is used. The user program is split into different parts, which execute in different tasks with typically different priorities. Normally the real time critical tasks (which require a certain reaction time) will have the highest priorities. **Multiple tasks** are used for the user interface, calling  $\mu$ C/GUI functions. These tasks typically have low priorities in the system, so they do not affect the real time behaviour of the system.

Interrupts can, but do not have to be used for real time parts of the software.

## Advantages

The real time behavior of the system is excellent. The real time behavior of a task is affected only by tasks running at higher priority. This means that changes of a program component running in a low priority task do not affect the real time behavior at all. If the user interface is executed from a low priority task, this means that changes on the user interface do not affect the real time behavior. This kind of system makes it easy to assign different components of the software to different members of the development team, which can work to a high degree independently from each other.

## Disadvantages

You have to have a real time kernel (RTOS), which costs money and uses up some ROM and RAM (for stacks). In addition, you will have to think about task synchronization and how to transfer information from one task to another.

## Using $\mu$ C/GUI

If the window manager's callback mechanism is used, then an  $\mu$ C/GUI update function (typically `GUI_Exec()`, `GUI_Delay()`) has to be called regularly from one or more tasks calling  $\mu$ C/GUI.

The default configuration, which does not support multitasking (`#define GUI_MT 0`) can **NOT** be used. The configuration needs to enable multitasking support and define a maximum number of tasks from which  $\mu$ C/GUI is called (excerpt from `GUIConf.h`):

```
#define GUI_MT 1           // Enable multitasking support
#define GUI_MAX_TASK 5     // Max. number of tasks that may call µC/GUI
```

Kernel interface routines are required, and need to match the kernel being used. You can use any real time kernel, commercial or proprietary. Both the macros and the routines are discussed in the following chapter sections.

## Recommendations

- Call the µC/GUI update functions (i.e. `GUI_Exec()`, `GUI_Delay()`) from just one task. It will help to keep the program structure clear. If you have sufficient RAM in your system, dedicate one task (with the lowest priority) to updating µC/GUI. This task will continuously call `GUI_Exec()` as shown in the example below and will do nothing else.
- Keep your real time tasks (which determine the behavior of your system with respect to I/O, interface, network, etc.) separate from tasks that call µC/GUI. This will help to assure best real time performance.
- If possible, use only one task for your user interface. This helps to keep the program structure simple and simplifies debugging. (However, this is not required and may not be suitable in some systems.)

## Example

This excerpt shows the dedicated µC/GUI update task. It is taken from the example `MT_Multitasking`, which is included in the samples shipped with µC/GUI:

```
/******
 *
 *           GUI background processing
 *
 * This task does the background processing.
 * The main job is to update invalid windows, but other things such as
 * evaluating mouse or touch input may also be done.
 */
void GUI_Task(void) {
    while(1) {
        GUI_Exec();           /* Do the background work ... Update windows etc. */
        GUI_X_ExecIdle();     /* Nothing left to do for the moment ... Idle processing */
    }
}µC/GUI
```

## 11.5 GUI configuration macros for multitasking support

The following table shows the configuration macros used for a multitask system with multiple tasks calling µC/GUI:

Type	Macro	Default	Explanation
N	GUI_MAXTASK	4	Define the maximum number of tasks from which µC/GUI is called when multitasking support is enabled (see below).
B	GUI_OS	0	Activate to enable multitasking support.

### GUI\_MAXTASK

#### Description

Defines the maximum number of tasks from which µC/GUI is called to access the display.

**Type**

Numerical value

**Additionalnnaal information**

This function is only relevant when GUI\_OS is activated.

**GUI\_OS**

**Description**

Enables multitasking support by activating the module GUITask.

**Type**

Binary switch

0: inactive, multitask support disabled (default)

1: active, multitask support enabled

**11.6 Kernel interface routine API**

An RTOS usually offers a mechanism called a resource semaphore, in which a task using a particular resource claims that resource before actually using it. The display is an example of a resource that needs to be protected with a resource semaphore. µC/GUI uses the macro GUI\_USE to call the function GUI\_Use() before it accesses the display or before it uses a critical internal data structure. In a similar way, it calls GUI\_Unuse() after accessing the display or using the data structure. This is done in the module GUITask.c.

GUITask.c in turn uses the GUI kernel interface routines shown in the table below. These routines are prefixed GUI\_X\_ since they are high-level (hardware-dependent) functions. They must be adapted to the real time kernel used in order to make the µC/GUI task (or thread) safe. Detailed descriptions of the routines follow, as well as examples of how they are adapted for different kernels.

Routine	Explanation
<a href="#">GUI_X_InitOS()</a>	Initialize the kernel interface module (create a resource semaphore/mutex).
<a href="#">GUI_X_GetTaskId()</a>	Return a unique, 32-bit identifier for the current task/thread.
<a href="#">GUI_X_Lock()</a>	Lock the GUI (block resource semaphore/mutex).
<a href="#">GUI_X_Unlock()</a>	Unlock the GUI (unblock resource semaphore/mutex).

**GUI\_X\_InitOS()**

**Description**

Creates the resource semaphore or mutex typically used by GUI\_X\_Lock() and GUI\_X\_Unlock().

**Prototype**

void GUI\_X\_InitOS(void)

**GUI\_X\_GetTaskID()**

**Description**

Returns a unique ID for the current task.

**Prototype**

```
U32 GUI_X_GetTaskID(void);
```

**Return value**

ID of the current task as a 32-bit integer.

**Additional information**

Used with a real-time operating system.

It does not matter which value is returned, as long as it is unique for each task/thread using the  $\mu$ C/GUI API and as long as the value is always the same for each particular thread.

**GUI\_X\_Lock()****Description**

Locks the GUI.

**Prototype**

```
void GUI_X_Lock(void);
```

**Additional information**

This routine is called by the GUI before it accesses the display or before using a critical internal data structure. It blocks other threads from entering the same critical section using a resource semaphore/mutex until `GUI_X_Unlock()` has been called.

When using a real time operating system, you normally have to increment a counting resource semaphore.

**GUI\_X\_Unlock()****Description**

Unlocks the GUI.

**Prototype**

```
void GUI_X_Unlock(void);
```

**Additional information**

This routine is called by the GUI after accessing the display or after using a critical internal data structure.

When using a real time operating system, you normally have to decrement a counting resource semaphore.

**Examples****Kernel interface routines for  $\mu$ C/OS-II**

The following example shows an adaption of the four routines for  $\mu$ C/OS-II (excerpt from file `GUI_X_uCOS-II.c`):

```
#include "INCLUDES.H"

static OS_EVENT * DispSem;

U32  GUI_X_GetTaskId(void) { return ((U32)(OSTCBCur->OSTCBPrio)); }
void GUI_X_InitOS(void)   { DispSem = OSemCreate(1); }
void GUI_X_Unlock(void)   { OSemPost(DispSem); }
void GUI_X_Lock(void)     { }
```

```

    INT8U err;
    OSSemPend(DispSem, 0, &err);
}

```

### Kernel interface routines for Win32

The following is an excerpt from the Win32 simulation for  $\mu$ C/GUI. (When using the  $\mu$ C/GUI simulation, there is no need to add these routines, as they are already in the library.)

Note: cleanup code has been omitted for clarity.

```

/*****
 *
 *       $\mu$ C/GUI - Multitask interface for Win32
 *
 *****/

The folling section consisting of 4 routines is used to make
 $\mu$ C/GUI thread safe with WIN32
*/

static HANDLE hMutex;

void GUI_X_InitOS(void) {
    hMutex = CreateMutex(NULL, 0, " $\mu$ C/GUISim - Mutex");
}

unsigned int GUI_X_GetTaskId(void) {
    return GetCurrentThreadId();
}

void GUI_X_Lock(void) {
    WaitForSingleObject(hMutex, INFINITE);
}

void GUI_X_Unlock(void) {
    ReleaseMutex(hMutex);
}

```

# Chapter 12

## The Window Manager (WM)

---

When using the  $\mu$ C/GUI window manager (WM), everything which appears on the display is contained in a window -- an area on the screen which acts as a user interface element for drawing or viewing objects. A window can be any size, and you can display multiple windows on the screen at once, even partially or entirely in front of other windows.

The window manager supplies a set of routines which allow you to easily create, move, resize, and otherwise manipulate any number of windows. It also provides lower-level support by managing the layering of windows on the display and by alerting your application to display changes that affect its windows.

The  $\mu$ C/GUI window manager is a separate (optional) software item and is not included in the  $\mu$ C/GUI basic package. The software for the window manager is located in the subdirectory `GUI\WM`.

## 12.1 Explanation of terms

Windows are rectangular in shape, defined by their origin (the X- and Y-coordinates of the upper left corner) as well as their X- and Y-sizes (width and height, respectively). A window in  $\mu$ C/GUI:

- is rectangular.
- has a Z-position.
- may be hidden or shown.
- may have valid and/or invalid areas.
- may or may not have transparency.
- may or may not have a callback routine.

### Active window

The window which is currently being used for drawing operations is referred to as the active window. It is not necessarily the same as the topmost window.

### Callback routines

Callback routines are defined by the user program, instructing the graphic system to call a specific function when a specific event occurs. Normally they are used to automatically redraw a window when its content has changed.

### Child/parent windows, siblings

A child window is one that is defined relative to another window, called the parent. Whenever a parent window moves, its child or children move correspondingly. A child window is always completely contained within its parent, and will be clipped if necessary. Multiple child windows with the same parent are considered "siblings" to one another.

### Client area

The client area of a window is simply its usable area. If a window contains a frame or title bar, then the client area is the rectangular inner area. If there is no such frame, then the coordinates of the client area are identical to those of the window itself.

### Clipping, clip area

Clipping is the process of limiting output to a window or part of it.

The clip area of a window is its visible area. This is the window area minus the area obstructed by siblings of higher Z-order, minus any part that does not fit into the visible area of the parent window.

### Desktop window

The desktop window is automatically created by the window manager, and always covers the entire display area. It is always the bottommost window, and when no other window has been defined, it is the default (active) window. All windows are descendants (children, grandchildren, etc.) of the desktop window.

### Handle

When a new window is created, the WM assigns it a unique identifier called a handle. The handle is used in any further operations performed on that particular window.

### Hiding/showing windows

A hidden window is not visible, although it still exists (has a handle). When a window is created, it is hidden by default if no create flag is specified. Showing a window makes it visible; hiding it makes it invisible.

## Transparency

A window that has transparency contains areas that are not redrawn with the rest of the window. These areas operate as though the window behind "shows through" them. In this case, it is important that the window behind is redrawn before the window with transparency. The WM automatically handles redrawing in the correct order.

## Validation/invalidation

A valid window is a fully updated window which does not need redrawing.

An invalid window does not yet reflect all updates and therefore needs to be redrawn, either completely or partially. When changes are made that affect a particular window, the WM marks that window as invalid. The next time the window is redrawn (either manually or by a callback routine) it will be validated.

## Z-position, bottom/top

Although a window is displayed on a two-dimensional screen in terms of X and Y, the WM also manages what is known as a Z-position, or depth coordinate -- a position in a virtual third dimension which determines its placement from background to foreground. Windows can therefore appear on top of or beneath one another.

Setting a window to the bottom will place it "underneath" all of its sibling windows (if any); setting it to the top will place it "on top of" its siblings. When a window is created, it is set to the top by default if no create flag is specified.

# 12.2 WM API

The following table lists the available routines of the  $\mu$ C/GUI window manager API. All functions are listed in alphabetical order within their respective categories. Detailed descriptions of the routines can be found later in the chapter.

Routine	Explanation
Basic functions	
<a href="#">WM_CreateWindow()</a>	Create a window.
<a href="#">WM_CreateWindowAsChild()</a>	Create a child window.
<a href="#">WM_DeleteWindow()</a>	Delete a window.
<a href="#">WM_Exec()</a>	Redraw invalid windows by executing callbacks (all jobs).
<a href="#">WM_Exec1()</a>	Redraw one invalid window by executing one callback (one job only).
<a href="#">WM_GetClientRect()</a>	Return the size of the active window.
<a href="#">WM_GetDialogItem()</a>	Return the window handle of a dialog box item (widget).
<a href="#">WM_GetOrgX()</a>	Return the origin in X of the active window.
<a href="#">WM_GetOrgY()</a>	Return the origin in Y of the active window.
<a href="#">WM_GetWindowOrgX()</a>	Return the origin in X of a window.
<a href="#">WM_GetWindowOrgY()</a>	Return the origin in Y of a window.
<a href="#">WM_GetWindowRect()</a>	Return the screen coordinates of the active window.
<a href="#">WM_GetWindowSizeX()</a>	Return the horizontal size (width) of a window.
<a href="#">WM_GetWindowSizeY()</a>	Return the vertical size (height) of a window.
<a href="#">WM_HideWindow()</a>	Make a window invisible.
<a href="#">WM_InvalidateArea()</a>	Invalidate a certain section of the display.
<a href="#">WM_InvalidateRect()</a>	Invalidate part of a window.
<a href="#">WM_InvalidateWindow()</a>	Invalidate a window.
<a href="#">WM_MoveTo()</a>	Set the position of a window.

Routine	Explanation
<code>WM_MoveWindow()</code>	Move a window to another position.
<code>WM_Paint()</code>	Draw or redraw a window immediately.
<code>WM_ResizeWindow()</code>	Change window size.
<code>WM_SelectWindow()</code>	Set the active window to be used for drawing operations.
<code>WM_ShowWindow()</code>	Make a window visible.
Advanced functions	
<code>WM_Activate()</code>	Activate the window manager.
<code>WM_BringToBottom()</code>	Place a window behind its siblings.
<code>WM_BringToTop()</code>	Place a window in front of its siblings.
<code>WM_ClrHasTrans()</code>	Clear the <code>has transparency</code> flag.
<code>WM_Deactivate()</code>	Deactivate the window manager.
<code>WM_DefaultProc()</code>	Default routine to handle messages.
<code>WM_GetActiveWindow()</code>	Return handle of the active window.
<code>WM_GetDesktopWindow()</code>	Return handle of the desktop window.
<code>WM_GetFirstChild()</code>	Return handle of a window's first child window.
<code>WM_GetNextSibling()</code>	Return handle of a window's next sibling.
<code>WM_GetHasTrans()</code>	Return current value of the <code>has transparency</code> flag.
<code>WM_GetParent()</code>	Return handle of a window's parent window.
<code>WM_Init()</code>	Initialize window manager. No longer necessary; done by <code>GUI_Init()</code> .
<code>WM_IsWindow()</code>	Determine whether a specified handle is a valid window handle.
<code>WM_SendMessage()</code>	Send a message to a window.
<code>WM_SetDesktopColor()</code>	Set desktop window color.
<code>WM_SetCallback()</code>	Set the callback routine for a window.
<code>WM_SetCreateFlags()</code>	Sets the flags to be used as default when creating new windows
<code>WM_SetHasTrans()</code>	Set the <code>has transparency</code> flag.
<code>WM_SetUserClipRect()</code>	Temporarily reduce the clipping area.
<code>WM_ValidateRect()</code>	Validate parts of a window.
<code>WM_ValidateWindow()</code>	Validate a window.
Memory device support (optional)	
<code>WM_DisableMemdev()</code>	Disable usage of memory devices for redrawing.
<code>WM_EnableMemdev()</code>	Enable usage of memory devices for redrawing.

## 12.3 Callback mechanism of the window manager

The WM may be used with or without callback routines. In most cases, using callbacks is preferable.

### Philosophy behind the callback mechanism

The idea behind the callback mechanism that  $\mu$ C/GUI offers for windows and window objects (widgets) is that of an event-driven system. As in most windowing systems, the principle is that the flow of control is not just from the user program to the graphic system, but also from the user program to the graphic system and back up to the user program by means of the callback routines provided by the user program. This mechanism -- often characterized as the Hollywood principle ("Don't call us,

we'll call you!") -- is needed by the window manager mainly in order to trigger the redrawing of windows. This contrasts with classical programming, but it makes it possible to exploit the invalidation logic of the window manager.

### Not using callbacks

You do not have to use callback routines, but in doing so, the WM loses the ability to manage redrawing (updating) of the windows. It is also possible to mix; for example, having some windows use callbacks and others not. However, if a window does not use the callback mechanism, your application is responsible for updating its contents.

**Warning: When not using the callback mechanism, it is your responsibility to manage screen updates!**

## 12.4 Using callback routines

In order to create a window with a callback, you must have a callback routine. The name of the routine will be the name used to specify the pointer when creating a window with that particular callback function (the `cb` parameter in `WM_CreateWindow()`). All callback routines must have the following prototype:

### Prototype

```
void callback(WM_MESSAGE* pMsg);
```

Parameter	Meaning
<code>pMsg</code>	Pointer to message.

The action performed by the callback routine depends on the type of message it receives. The prototype above is usually followed by a `switch` statement, which defines different behaviors for different messages using one or more `case` statements (typically at least `WM_PAINT()`).

### Example

Creates a callback routine to automatically redraw a window:

```
void WinHandler(WM_MESSAGE* pMsg) {
    switch (pMsg->MsgId) {
        case WM_PAINT:
            GUI_SetBkColor(0xFF00);
            GUI_Clear();
            GUI_DispStringAt("Hello world",0,0);
            break;
    }
}
```

## Elements of WM\_MESSAGE

MsgId	Type of message (see table below).
hWin	Destination window.
hWinSrc	Source window.
Data.p	Data pointer.
Data.v	Data value.

### Types of messages used with MsgId

WM_PAINT	Redraws the window (because content is at least partially invalid).
WM_CREATE	Sent immediately after a window has been created.
WM_DELETE	Tells the window to free its data structures (if any) since it is about to be deleted.
WM_SIZE	Sent to a window when its size has changed.
WM_MOVE	Sent to a window when it has been moved.
WM_SHOW	Sent to a window when it has received the show command.
WM_HIDE	Sent to a window when it has received the hide command.
WM_TOUCH	Touch-screen message.

The application program can define additional messages for its own usage. In order to ensure that they do not use the same message ID's as those used by  $\mu$ C/GUI, user-defined messages start numbering after WM\_USER. You would define your own messages as follows:

```
#define MY_MESSAGE_AAA WM_USER+0
#define MY_MESSAGE_BBB WM_USER+1
and so on.
```

### Background window redrawal and callback

During initialization of the window manager, a window containing the whole LCD area is created as a background window. The handle of this window is WM\_HBKWIN. The WM does not redraw areas of the background window automatically, because there is no default background color. That means if you create a further window and then delete it, the deleted window will still be visible. The routine WM\_SetBkWindowColor() needs to be specified in order to set a color for redrawing the background window.

You can also set a callback function to take care of this problem. If a window is created and then deleted as before, the callback routine will trigger the WM to recognize that the background window is no longer valid and redraw it automatically. For more information on creating and using callback routines, see the example at the end of the chapter.

## 12.5 Basic functions

### WM\_CreateWindow()

#### Description

Creates a window of a specified size at a specified location.

#### Prototype

```
WM_HWIN WM_CreateWindow(int x0, int y0,
                        int width, int height,
```

```

U8 Style,
WM_CALLBACK* cb
    int NumExtraBytes);

```

Parameter	Meaning
x0	Upper left X-position.
y0	Upper left Y-position.
width	X-size of window.
height	Y-size of window.
Style	Window create flags, listed below.
cb	Pointer to callback routine, or NULL if no callback used.
NumExtra-Bytes	Number of extra bytes to be allocated, normally 0.

Permitted values for parameter <code>Style</code> (OR-combinable)	
WM_CF_HASTRANS	Has transparency flag. Must be defined for windows whose client area is not entirely filled.
WM_CF_HIDE	Hide window after creation (default).
WM_CF_SHOW	Show window after creation.
WM_CF_FGND	Put window in foreground after creation (default).
WM_CF_BGND	Put window in background after creation.
WM_CF_STAYONTOP	Make sure window stays on top of all siblings created without this flag.
WM_CF_MEMDEV	Automatically use a memory device when redrawing. This will avoid flicker and also improve the output speed in most cases, as clipping is simplified. Note that the memory device package is required (and needs to be enabled in the configuration) in order to be able to use this flag. If memory devices are not enabled, this flag is ignored.

## Return value

Handle for created window.

## Additional information

Several create flags can be combined by using the (OR) operator. Negative-position coordinates may be used.

## Examples

Creates a window with callback:

```
hWin2 = WM_CreateWindow(100, 10, 180, 100, WM_CF_SHOW, &WinHandler, 0);
```

Creates a window without callback:

```
hWin2 = WM_CreateWindow(100, 10, 180, 100, WM_CF_SHOW, NULL, 0);
```

## WM\_CreateWindowAsChild()

### Description

Creates a window as a child window.

### Prototype

```

WM_HWIN WM_CreateWindowAsChild(int x0, int y0,
                                int width, int height,
                                WM_HWIN hWinParent,

```

```
U8 Style,
WM_CALLBACK* cb
    int NumExtraBytes);
```

Parameter	Meaning
<code>x0</code>	Upper left X-position relative to parent window.
<code>y0</code>	Upper left Y-position relative to parent window.
<code>width</code>	X-size of window. If 0, X-size of client area of parent window.
<code>height</code>	Y-size of window. If 0, Y-size of client area of parent window.
<code>hWinParent</code>	Handle of parent window.
<code>Style</code>	Window create flags (see <code>WM_CreateWindow()</code> ).
<code>cb</code>	Pointer to callback routine, or NULL if no callback used.
<code>NumExtra-Bytes</code>	Number of extra bytes to be allocated, normally 0.

### Return value

Handle for the child window.

### Additional information

If the `hWinParent` parameter is set to 0, the background window is used as parent. A child window is placed on top of its parent and any previous siblings by default, so that if their Z-positions are not changed, the "youngest" window will always be top-most.

The Z-positions of siblings may be changed, although they will always remain on top of their parent regardless of their order.

## WM\_DeleteWindow()

### Description

Deletes a specified window.

### Prototype

```
void WM_DeleteWindow(WM_HWIN hWin);
```

Parameter	Meaning
<code>hWin</code>	Window handle.

### Additional information

Before the window is deleted, it receives a `WM_DELETE` message. This message is typically used to delete any objects (widgets) it uses and to free memory dynamically allocated by the window.

If the specified window has any existing child windows, these are automatically deleted before the window itself is deleted.

## WM\_Exec()

### Description

Redraws invalid windows by executing callback functions (all jobs).

### Prototype

```
int WM_Exec(void);
```

**Return value**

0 if there were no jobs performed.

1 if a job was performed.

**Additional information**

This function will automatically call `WM_Exec1()` repeatedly until it has completed all jobs -- essentially until a 0 value is returned.

It is recommended to call the function `GUI_Exec()` instead.

Normally this function does not need to be called by the user application. It is called automatically by `GUI_Delay()`. If you are using a multitasking system, we recommend executing this function by a separate task as seen below:

```
void ExecIdleTask(void) {
    while(1) {
        WM_Exec();
    }
}
```

**WM\_Exec1()****Description**

Redraws an invalid window by executing one callback function (one job only).

**Prototype**

```
int WM_Exec1(void);
```

**Return value**

0 if there were no jobs performed.

1 if a job was performed.

**Additional information**

This routine may be called repeatedly until 0 is returned, which means all jobs have been completed.

It is recommended to call the function `GUI_Exec1()` instead.

This function is called automatically by `WM_Exec()`.

**WM\_GetClientRect()****Description**

Returns the coordinates of the client area in the active window.

**Prototype**

```
void WM_GetClientRect(GUI_RECT* pRect);
```

Parameter	Meaning
<code>pRect</code>	Pointer to a <code>GUI_RECT</code> structure.

**WM\_GetDialogItem()****Description**

Returns the window handle of a dialog box item (widget).

## Prototype

```
WM_HWIN WM_GetDialogItem(WM_HWIN hDialog, int Id);
```

Parameter	Meaning
<a href="#">hDialog</a>	Handle of dialog box.
<a href="#">Id</a>	Window ID of the widget.

## Return value

The window handle of the widget.

## Additional information

This function is always used when creating dialog boxes, since the window ID of a widget used in a dialog must be converted to its handle before it can be used.

## WM\_GetOrgX(), WM\_GetOrgY()

### Description

Return the X- or Y-position (respectively) of the origin of the client area in the active window.

### Prototypes

```
int WM_GetOrgX(void);  
int WM_GetOrgY(void);
```

### Return value

X- or Y-position of the client area in pixels.

## WM\_GetWindowOrgX(), WM\_GetWindowOrgY()

### Description

Return the X- or Y-position (respectively) of the origin of the client area in a specified window.

### Prototypes

```
int WM_GetWindowOrgX(WM_HWIN hWin);  
int WM_GetWindowOrgY(WM_HWIN hWin);
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.

### Return value

X- or Y-position of the client area in pixels.

## WM\_GetWindowRect()

### Description

Returns the coordinates of the active window.

## Prototype

```
void WM_GetWindowRect(GUI_RECT* pRect);
```

Parameter	Meaning
<a href="#">pRect</a>	Pointer to a GUI_RECT structure.

## WM\_GetWindowSizeX(), WM\_GetWindowSizeY()

### Description

Return the X- or Y-size (respectively) of a specified window.

### Prototypes

```
int WM_GetWindowSizeX(WM_HWIN hWin);
```

```
int WM_GetWindowSizeY(WM_HWIN hWin);
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.

### Return value

X- or Y-size of the window in pixels.

Defined as  $x1 - x0 + 1$  in horizontal direction,  $y1 - y0 + 1$  in vertical direction, where  $x0$ ,  $x1$ ,  $y0$ ,  $y1$  are the leftmost/rightmost/topmost/bottommost positions of the window.

## WM\_HideWindow()

### Description

Makes a specified window invisible.

### Prototype

```
void WM_HideWindow(WM_HWIN hWin);
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.

### Additional information

The window will not immediately appear "invisible" after calling this function. The invalid areas of other windows (areas which appear to lie "behind" the window which should be hidden) will be redrawn when executing `WM_Exec()`. If you need to hide (draw over) a window immediately, you should call `WM_Paint()` to redraw the other windows.

## WM\_InvalidateArea()

### Description

Invalidates a specified, rectangular area of the display.

## Prototype

```
void WM_InvalidateArea(GUI_RECT* pRect);
```

Parameter	Meaning
<a href="#">pRect</a>	Pointer to a GUI_RECT structure.

## Additional information

Calling this function will tell the WM that the specified area is not updated. This function can be used to invalidate any windows or parts of windows that overlap or intersect the area.

## WM\_InvalidateRect()

### Description

Invalidates a specified, rectangular area of a window.

### Prototype

```
void WM_InvalidateRect(WM_HWIN hWin, GUI_RECT* pRect);
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.
<a href="#">pRect</a>	Pointer to a GUI_RECT structure.

## Additional information

Calling this function will tell the WM that the specified area is not updated. The next time WM\_Paint() is called to redraw the window, the area will be redrawn as well.

## WM\_InvalidateWindow()

### Description

Invalidates a specified window.

### Prototype

```
void WM_InvalidateWindow(WM_HWIN hWin);
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.

## Additional information

Calling this function will tell the WM that the specified window is not updated.

## WM\_MoveTo()

### Description

Moves a specified window to a certain position.

## Prototype

```
void WM_MoveTo(WM_HWIN hWin, int dx, int dy);
```

Parameter	Meaning
<code>hWin</code>	Window handle.
<code>x</code>	New X-position.
<code>y</code>	New Y-position.

## WM\_MoveWindow()

### Description

Moves a specified window by a certain distance.

### Prototype

```
void WM_MoveWindow(WM_HWIN hWin, int dx, int dy);
```

Parameter	Meaning
<code>hWin</code>	Window handle.
<code>dx</code>	Horizontal distance to move.
<code>dy</code>	Vertical distance to move.

## WM\_Paint()

### Description

Draws or redraws a specified window immediately.

### Prototype

```
void WM_Paint(WM_HWIN hWin);
```

Parameter	Meaning
<code>hWin</code>	Window handle.

### Additional information

The window is redrawn reflecting all updates made since the last time it was drawn.

## WM\_ResizeWindow()

### Description

Changes the size of a specified window.

### Prototype

```
void WM_ResizeWindow(WM_HWIN hWin, int XSize, int YSize);
```

Parameter	Meaning
<code>hWin</code>	Window handle.
<code>XSize</code>	X-size to change window to.
<code>YSize</code>	Y-size to change window to.

## WM\_SelectWindow()

### Description

Sets the active window to be used for drawing operations.

### Prototype

```
WM_HWIN WM_SelectWindow(WM_HWIN hWin);
```

Parameter	Meaning
<code>hWin</code>	Window handle.

### Return value

The selected window.

### Example

Sets a window with handle `hWin2` to the active window, sets the background color, and then clears the window:

```
WM_SelectWindow(hWin2);  
GUI_SetBkColor(0xFF00);  
GUI_Clear();
```

## WM\_ShowWindow()

### Description

Makes a specified window visible.

### Prototype

```
void WM_ShowWindow(WM_HWIN hWin);
```

Parameter	Meaning
<code>hWin</code>	Window handle.

### Additional information

The window will not immediately be visible after calling this function. It will be redrawn when executing `WM_Exec()`. If you need to show (draw) the window immediately, you should call `WM_Paint()`.

## 12.6 Advanced functions

### WM\_Activate()

#### Description

Activates the window manager.

#### Prototype

```
void WM_Activate(void);
```

#### Additional information

The WM is activated by default after initialization. This function only needs to be called if there has been a previous call of `WM_Deactivate()`.

## WM\_BringToBottom()

### Description

Places a specified window underneath its siblings.

### Prototype

```
void WM_BringToBottom(WM_HWIN hWin);
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.

### Additional information

The window will be placed underneath all other sibling windows, but will remain in front of its parent.

## WM\_BringToTop()

### Description

Places a specified window on top of its siblings.

### Prototype

```
void WM_BringToTop(WM_HWIN hWin);
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.

### Additional information

The window will be placed on top of all other sibling windows and its parent.

## WM\_ClrHasTrans()

### Description

Clears the `has transparency` flag (sets it to 0).

### Prototype

```
void WM_ClrHasTrans(WM_HWIN hWin);
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.

### Additional information

When set, this flag tells the window manager that a window contains sections which are not redrawn and will therefore be transparent. The WM then knows that the background needs to be redrawn prior to redrawing the window in order to make sure the transparent sections are restored correctly.

When the flag is cleared with `WM_ClrHasTrans()`, the WM will not automatically redraw the background before redrawing the window.

## WM\_Deactivate()

### Description

Deactivates the window manager.

### Prototype

```
void WM_Deactivate(void);
```

### Additional information

After calling this function, the clip area is set to the complete LCD area and the WM will not execute window callback functions.

## WM\_DefaultProc()

### Description

Default message handler.

### Prototype

```
void WM_DefaultProc(WM_MESSAGE* pMsg)
```

Parameter	Meaning
<a href="#">pMsg</a>	Pointer to message.

### Additional information

Use this function to handle unprocessed messages as in the following example:

```
static WM_RESULT cbBackgroundWin(WM_MESSAGE* pMsg) {  
    switch (pMsg->MsgId) {  
        case WM_PAINT:  
            GUI_Clear();  
        default:  
            WM_DefaultProc(pMsg);  
    }  
}
```

## WM\_GetActiveWindow()

### Description

Returns the handle of the active window used for drawing operations.

### Prototype

```
WM_HWIN WM_GetActiveWindow(void);
```

### Return value

The handle of the active window.

## WM\_GetDesktopWindow()

### Description

Returns the handle of the desktop window.

### Prototype

```
WM_HWIN WM_GetDesktopWindow(void);
```

**Return value**

The handle of the desktop window.

**Additional information**

The desktop window is always the bottommost window and any further created windows are its descendants.

**WM\_GetFirstChild()****Description**

Returns the handle of a specified window's first child window.

**Prototype**

```
void WM_GetFirstChild(WM_HWIN hWin);
```

Parameter	Meaning
<code>hWin</code>	Window handle.

**Return value**

Handle of the window's first child window; 0 if no child window exists.

**Additional information**

A window's first child window is the first child created to that particular parent. If the Z-positions of the windows have not been changed, it will be the window directly on top of the specified parent.

**WM\_GetNextSibling()****Description**

Returns the handle of a specified window's next sibling.

**Prototype**

```
void WM_GetNextSibling(WM_HWIN hWin);
```

Parameter	Meaning
<code>hWin</code>	Window handle.

**Return value**

Handle of the window's next sibling; 0 if none exists.

**Additional information**

A window's next sibling is the next child window that was created relative to the same parent. If the Z-positions of the windows have not been changed, it will be the window directly on top of the one specified.

**WM\_GetHasTrans()****Description**

Returns the current value of the `has transparency` flag.

## Prototype

```
U8 WM_GetHasTrans(WM_HWIN hWin);
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.

## Return value

0: no transparency

1: window has transparency

## Additional information

When set, this flag tells the window manager that a window contains sections which are not redrawn and will therefore be transparent. The WM then knows that the background needs to be redrawn prior to redrawing the window in order to make sure the transparent sections are restored correctly.

## WM\_GetParent()

### Description

Returns the handle of a specified window's parent window.

### Prototype

```
void WM_GetParent(WM_HWIN hWin);
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.

## Return value

Handle of the window's parent window; 0 if none exists.

## Additional information

The only case in which no parent window exists is if the handle of the desktop window is used as parameter.

## WM\_Init()

### Description

Initializes the window manager.

### Prototype

```
void WM_Init(void);
```

## Additional information

It is no longer necessary to call this function by the user application. It is called by `GUI_Init()`.

## WM\_IsWindow()

### Description

Determines whether or not a specified handle is a valid window handle.

## Prototype

```
void WM_IsWindow(WM_HWIN hWin);
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.

## Return value

0: handle is not a valid window handle  
 1: handle is a valid window handle

## WM\_SendMessage()

### Description

Sends a message to a specified window.

### Prototype

```
void WM_SendMessage(WM_HWIN hWin, WM_MESSAGE* pMsg)
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.
<a href="#">pMsg</a>	Pointer to message.

## WM\_SetDesktopColor()

### Description

Sets the color for the desktop window.

### Prototype

```
GUI_COLOR WM_SetDesktopColor(GUI_COLOR Color);
```

Parameter	Meaning
<a href="#">Color</a>	Color for desktop window, 24-bit RGB value.

## Return value

The previously selected desktop window color.

## Additional information

The default setting for the desktop window is not to repaint itself. If this function is not called, the desktop window will not be redrawn at all; therefore other windows will remain visible even after they are deleted.

Once a color is specified with this function, the desktop window will repaint itself. In order to restore the default, call this function and specify `GUI_INVALID_COLOR`.

## WM\_SetCallback()

### Description

Sets a callback routine to be executed by the window manager.

## Prototype

```
WM_CALLBACK* WM_SetCallback (WM_HWIN hWin, WM_CALLBACK* cb)
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.
<a href="#">cb</a>	Pointer to callback routine.

## Return value

Pointer to the previous callback routine.

## WM\_SetCreateFlags()

### Description

Sets the flags to be used as default when creating a new window.

### Prototype

```
U8 WM_SetCreateFlags (U8 Flags)
```

Parameter	Meaning
<a href="#">Flags</a>	Window create flags (see <code>WM_CreateWindow()</code> ).

## Return value

Former value of this parameter.

### Additional information

The flags specified here are binary `ORed` with the flags specified in the `WM_CreateWindow()` and `WM_CreateWindowAsChild()` routines.

The flag `WM_CF_MEMDEV` is frequently used to enable memory devices on all windows.

### Example

```
WM_SetCreateFlags(WM_CF_MEMDEV); /* Auto. use memory devices on all windows */
```

## WM\_SetHasTrans()

### Description

Sets the `has transparency` flag (sets it to 1).

### Prototype

```
void WM_SetHasTrans(WM_HWIN hWin);
```

Parameter	Meaning
<a href="#">HWin</a>	Window handle.

### Additional information

When set, this flag tells the window manager that a window contains sections which are not redrawn and will therefore be transparent. The WM then knows that the background needs to be redrawn prior to redrawing the window in order to make sure the transparent sections are restored correctly.

## WM\_SetUserClipRect()

### Description

Temporarily reduces the clip area of the current window to a specified rectangle.

### Prototype

```
const GUI_RECT* WM_SetUserClipRect(const GUI_RECT* pRect);
```

Parameter	Meaning
<code>pRect</code>	Pointer to a GUI_RECT structure defining the clipping region.

### Return value

Pointer to the previous clip rectangle.

### Additional information

A NULL pointer can be passed in order to restore the default settings. The clip rectangle will automatically be reset by the WM when callbacks are used.

The given rectangle must be relative to the current window. You can not enlarge the clip rectangle beyond the current window rectangle.

Your application must ensure that the specified rectangle retains its value until it is no longer needed; i.e. until a different clip rectangle is specified or until a NULL pointer is passed. This means that the rectangle structure passed as parameter should not be an auto variable (usually located on the stack) if the clip rectangle remains active until after the return. In this case, a static variable should be used.

### Example

This example is taken from the drawing routine of a progress indicator. The progress indicator must write text on top of the progress bar, where the text color has to be different on the left and right parts of the bar. This means that half of a digit could be in one color, while the other half could be in a different color. The best way to do this is to temporarily reduce the clip area when drawing each part of the bar as shown below:

```
/* Draw left part of the bar */
r.x0=0; r.x1=x1-1; r.y0=0; r.y1 = GUI_YMAX;
WM_SetUserClipRect(&r);
GUI_SetBkColor(pThis->ColorBar[0]);
GUI_SetColor(pThis->ColorText[0]);
GUI_Clear();
GUI_GotoXY(xText,yText); GUI_DispDecMin(pThis->v); GUI_DispChar('%');
/* Draw right part of the bar */
r.x0=r.x1; r.x1=GUI_XMAX;
WM_SetUserClipRect(&r);
GUI_SetBkColor(pThis->ColorBar[1]);
GUI_SetColor(pThis->ColorText[1]);
GUI_Clear();
GUI_GotoXY(xText,yText); GUI_DispDecMin(pThis->v); GUI_DispChar('%');
```

### Screen shot of progress bar



## WM\_ValidateRect()

### Description

Validates a specified, rectangular area of a window.

## Prototype

```
void WM_ValidateRect (WM_HWIN hWin, GUI_RECT* pRect);
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.
<a href="#">pRect</a>	Pointer to a GUI_RECT structure.

## Additional information

Calling this function will tell the WM that the specified area is updated. Normally this function is called internally and does not need to be called by the user application.

## WM\_ValidateWindow()

### Description

Validates a specified window.

### Prototype

```
void WM_ValidateWindow (WM_HWIN hWin);
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.

## Additional information

Calling this function will tell the WM that the specified window is updated. Normally this function is called internally and does not need to be called by the user application.

## 12.7 Memory device support (optional)

When a memory device is used for redrawing a window, all drawing operations are automatically routed to a memory device context and are executed in memory. Only after all drawing operations have been carried out is the window redrawn on the LCD, reflecting all updates at once. The advantage of using memory devices is that any flickering effects (which normally occur when the screen is continuously updated as drawing operations are executed) are eliminated.

For more information on how memory devices operate, see Chapter 10: "Memory Devices".

## WM\_DisableMemdev()

### Description

Disables the use of memory devices for redrawing a window.

### Prototype

```
void WM_EnableMemdev (WM_HWIN hWin)
```

Parameter	Meaning
<a href="#">hWin</a>	Window handle.

## WM\_EnableMemdev()

### Description

Enables the use of memory devices for redrawing a window.

### Prototype

```
void WM_EnableMemdev (WM_HWIN hWin)
```

Parameter	Meaning
<code>hWin</code>	Window handle.

## 12.8 Example

The following example illustrates the difference between using a callback routine for redrawing the background and not having one. It also shows how to set your own callback function. The example is available as `WM_Redraw.c` in the samples shipped with  $\mu$ C/GUI:

```

/*****
*                               Micrium Inc.                               *
*                               Empowering embedded systems               *
*                               uC/GUI sample code                       *
*                               ****                                     ****
*****/

-----
File      : WM_Redraw.c
Purpose   : Demonstrates the redrawing mechanism of the window manager
-----
*/

#include "GUI.H"

/*****
*                               Callback routine for background window      *
*                               ****                                     ****
*****/

static void cbBackgroundWin(WM_MESSAGE* pMsg) {
    switch (pMsg->MsgId) {
        case WM_PAINT:
            GUI_Clear();
        default:
            WM_DefaultProc(pMsg);
    }
}

/*****
*                               Callback routine for foreground window      *
*                               ****                                     ****
*****/

static void cbForegroundWin(WM_MESSAGE* pMsg) {
    switch (pMsg->MsgId) {
        case WM_PAINT:
            GUI_SetBkColor(GUI_GREEN);
            GUI_Clear();
            GUI_DispString("Foreground window");
            break;
    }
}

```

```

    default:
        WM_DefaultProc(pMsg);
    }
}

/*****
*
*       Demonstrates the redraw mechanism of µC/GUI
*
*****/

static void DemoRedraw(void) {
    GUI_HWIN hWnd;
    while(1) {
        /* Create foreground window */
        hWnd = WM_CreateWindow(10, 10, 100, 100, WM_CF_SHOW, cbForegroundWin, 0);
        /* Show foreground window */
        GUI_Delay(1000);
        /* Delete foreground window */
        WM_DeleteWindow(hWnd);
        GUI_DispatchStringAt("Background of window has not been redrawn", 10, 10);
        /* Wait a while, background will not be redrawn */
        GUI_Delay(1000);
        GUI_Clear();
        /* Set callback for Background window */
        WM_SetCallback(WM_HBKWIN, cbBackgroundWin);
        /* Create foreground window */
        hWnd = WM_CreateWindow(10, 10, 100, 100, WM_CF_SHOW, cbForegroundWin, 0);
        /* Show foreground window */
        GUI_Delay(1000);
        /* Delete foreground window */
        WM_DeleteWindow(hWnd);
        /* Wait a while, background will be redrawn */
        GUI_Delay(1000);
        /* Delete callback for Background window */
        WM_SetCallback(WM_HBKWIN, 0);
    }
}

/*****
*
*       main
*
*****/

void main(void) {
    GUI_Init();
    DemoRedraw();
}

```

# Chapter 13

## Window Objects (Widgets)

---

Widgets are windows with object-type properties; they are called controls in the windows world and make up the elements of the user interface. They can react automatically to certain events; for example, a button can appear in a different state if it is pressed. Widgets need to be created, have properties which may be changed at any time during their existence and are then typically deleted when they are no longer needed. Just like a window, a widget is referenced by a handle which is returned by its create function.

Widgets require the window manager. Once a widget is created, it is treated just like any other window; the WM ensures that it is properly displayed (and redrawn) whenever necessary. Widgets are not required when writing an application or a user interface, but they can make programming much easier.

## 13.1 Some basics

### Available widgets

The following  $\mu$ C/GUI widgets are currently available:

Name	Explanation
BUTTON	Button which can be pressed. Text or bitmaps may be displayed on a button.
CHECKBOX	Check box which may be checked or unchecked.
EDIT	Single-line edit field which prompts the user to type a number or text.
FRAMEWIN	Frame window. Creates the typical GUI look.
LISTBOX	Listbox which highlights items as they are selected by the user.
PROGBAR	Progress bar used for visualization.
RADIOBUTTON	Radio button which may be selected. Only one button may be selected at a time.
SCROLLBAR	Scrollbar which may be horizontal or vertical.
SLIDER	Slider bar used for changing values.
TEXT	Static text controls typically used in dialogs.

### Understanding the redrawing mechanism

A widget draws itself according to its properties. This is done when `WM_Exec()` is called. If you do not call `WM_Exec()` from within your program, `WM_Paint()` must be called for the widget.  $\mu$ C/GUI in a multitasking environment, a background task is normally used to call `WM_Exec()` and update the widgets (and all other windows with callback functions). It is then not necessary to manually call `WM_Paint()`; however, it is still legal to do so and may also make sense if you want to ensure that the widget is redrawn immediately.

When a property of a widget is changed, the window of the widget (or part of it) is marked as invalid, but it is not immediately redrawn. Therefore, the section of code executes very fast. The redrawal is done by the WM at a later time or it can be forced by calling `WM_Paint()` for the widget (or `WM_Exec()` until all windows are redrawn).

### How to use widgets

Suppose we would like to display a progress bar. All that is needed is the following code:

```
PROGBAR_Handle hProgBar;
GUI_DispStringAt("Progress bar", 100, 20);
hProgBar = PROGBAR_Create(100, 40, 100, 20, WM_CF_SHOW);
```



The first line reserves memory for the handle of the widget. The last line actually creates the widget. The widget will then automatically be drawn by the window manager if `WM_Exec()` is called at a later point or in a separate task.

Member functions are available for each type of widget which allow modifications to their appearance. Once the widget has been created, its properties can be changed by calling one of its member functions. These functions take the handle of the widget as their first argument. In order to make the progress bar created above show 45% and to change the bar colors from their defaults (dark gray/light gray) to green/red, the following code section may be used:

```
PROGBAR_SetBarColor(hProgBar, 0, GUI_GREEN);
PROGBAR_SetBarColor(hProgBar, 1, GUI_RED);
PROGBAR_SetValue(hProgBar, 45);
```



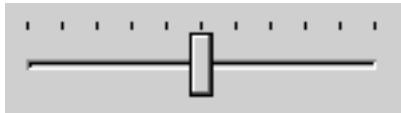
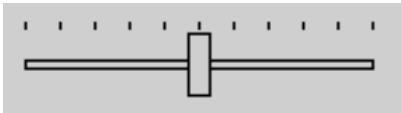
All widgets also have one or more configuration macros which define various default settings such as fonts and colors used. The available configuration options are listed for each widget in its respective section later in the chapter.

### Dynamic memory usage for widgets

In embedded applications it is usually not very desirable to use dynamic memory at all because of fragmentation effects. There are a number of different strategies that can be used to avoid this, but they all work in a limited way whenever memory areas are referenced by a pointer in the application program. For this reason,  $\mu$ C/GUI uses a different approach: all objects (and all data stored at run-time) are stored in memory areas referenced by a handle. This makes it possible to relocate the allocated memory areas at run-time, thus avoiding the long-term allocation problems which occur when using pointers. All widgets are thus referenced by handles.

### 3D support

Many widgets may be displayed with or without 3D effects. 3D support is enabled by default, but may be disabled by setting the configuration macro `<WIDGET>_USE_3D` to 0. A widget will function exactly the same way whether it uses three-dimensional effects or not; the only difference will be in its appearance. This is demonstrated below with a slider widget:

3D effects enabled (default)	3D effects disabled
	

## 13.2 General widget API

### API reference: WM routines used for widgets

Since widgets are essentially windows, they are compatible with any of the window manager API routines. The handle of the widget is used as the `hWin` parameter and the widget is treated like any other window. The WM functions most commonly used with widgets are listed as follows:

Routine	Explanation
<code>WM_DeleteWindow()</code>	Delete a window.
<code>WM_DisableMemdev()</code>	Disable usage of memory devices for redrawing.
<code>WM_EnableMemdev()</code>	Enable usage of memory devices for redrawing.
<code>WM_InvalidateWindow()</code>	Invalidate a window.
<code>WM_Paint()</code>	Draw or redraw a window immediately.

For a complete list of WM-related functions, please refer to Chapter 12: "The Window Manager".

## API reference: routines common to all widgets

The table below lists available widget-related routines in alphabetical order. These functions are common to all widgets, and are listed here in order to avoid repetition. Detailed descriptions of the routines follow. The additional member functions available for each widget may be found in later sections.

Routine	Explanation
<code>&lt;WIDGET&gt;_CreateIndirect()</code>	Used for automatic creation in dialog boxes.
<code>WM_EnableWindow()</code>	Set the widget state to enabled (default).
<code>WM_DisableWindow()</code>	Set the widget state to disabled.

### <WIDGET>\_CreateIndirect()

#### Description

Creates a widget to be used in dialog boxes.

#### Prototype

```
<WIDGET>_Handle <WIDGET>_CreateIndirect(const GUI_WIDGET_CREATE_INFO*
                                         pCreateInfo, WM_HWIN hParent,
                                         int x0, int y0, WM_CALLBACK* cb);
```

Parameter	Meaning
<code>pCreateInfo</code>	Pointer to a GUI_WIDGET_CREATE_INFO structure (see below).
<code>hParent</code>	Handle of parent window.
<code>x0</code>	Leftmost pixel of the widget (in desktop coordinates).
<code>y0</code>	Topmost pixel of the widget (in desktop coordinates).
<code>cb</code>	Pointer to a callback funtion.

#### Additional information

Any widget may be created indirectly by using the appropriate prefix. For example: `BUTTON_CreateIndirect()` to indirectly create a button widget, `CHECKBOX_CreateIndirect()` to indirectly create a check box widget, and so on.

A widget only needs to be created indirectly if it is to be included in a dialog box. Otherwise, it may be created directly by using the `<WIDGET>_Create()` functions. Please see Chapter 14: "Dialogs" for more information about dialog boxes.

#### Resource table

The GUI\_WIDGET\_CREATE\_INFO structure is defined in the dialog resource table as follows:

```
typedef struct {
    GUI_WIDGET_CREATE_FUNC* pfCreateIndirect; // Create function
    const char* pName;                        // Text (not used for all widgets)
    I16 Id;                                    // Window ID of the widget
    I16 x0, y0, xSize, ySize;                 // Size and position of the widget
    I16 Flags;                                // Widget-specific flags (or 0)
    I32 Para;                                  // Widget-specific parameter (or 0)
} GUI_WIDGET_CREATE_INFO;
```

Widget flags and parameters are optional, and vary depending on the type of widget. The available flags and parameters for each widget (if any) will be listed under the appropriate section later in the chapter.

## WM\_EnableWindow()

### Description

Set a specified widget to an active (usable) state.

### Prototype

```
void WM_EnableWindow(WM_Handle hObj);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of widget.

### Additional information

This is the default setting for any widget.

## WM\_DisableWindow()

### Description

Set a specified widget to an inactive (non-usable) state.

### Prototype

```
void WM_DisableWindow(WM_Handle hObj);
```

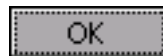
Parameter	Meaning
<a href="#">hObj</a>	Handle of widget.

### Additional information

A widget that is disabled will typically appear gray, and will not accept input from the user. However, the actual appearance may vary (depends on widget/configuration settings, etc.)

## 13.3 BUTTON: Button widget

Button widgets are commonly used as the primary user interface element for touchscreens. Buttons may be displayed with text, as shown below, or with a bitmap.



All BUTTON-related routines are located in the file(s) `BUTTON*.c`, `BUTTON.h`. All identifiers are prefixed `BUTTON`.

## Configuration options

Type	Macro	Default	Explanation
N	BUTTON_3D_MOVE_X	1	Number of pixels that text/bitmap moves in horizontal direction in pressed state.
N	BUTTON_3D_MOVE_Y	1	Number of pixels that text/bitmap moves in vertical direction in pressed state.
N	BUTTON_BKCOLOR0_DEFAULT	0xAAAAAA	Background color, unpressed state.
N	BUTTON_BKCOLOR1_DEFAULT	GUI_WHITE	Background color, pressed state.
S	BUTTON_FONT_DEFAULT	&GUI_Font13_1	Font used for button text.
N	BUTTON_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color, unpressed state.
N	BUTTON_TEXTCOLOR1_DEFAULT	GUI_BLACK	Text color, pressed state.
B	BUTTON_USE_3D	1	Enable support for 3D effects.

The default for the button is to use a white background in pressed state. This has been done purposely because it makes it very obvious that the button is pressed, on any kind of display. If you want the background color of the button to be the same in both its pressed and unpressed states, change `BUTTON_BKCOLOR1_DEFAULT` to `BUTTON_BKCOLOR0_DEFAULT`.

## BUTTON API

The table below lists the available  $\mu$ C/GUI BUTTON-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<code>BUTTON_Create()</code>	Create the button.
<code>BUTTON_CreateAsChild()</code>	Create the button as a child window.
<code>BUTTON_CreateIndirect()</code>	Create the button from resource table entry.
<code>BUTTON_SetBitmap()</code>	Set the bitmap used when displaying the button.
<code>BUTTON_SetBitmapEx()</code>	Set the bitmap used when displaying the button.
<code>BUTTON_SetBkColor()</code>	Set the background color of the button.
<code>BUTTON_SetFont()</code>	Select the font for the text.
<code>BUTTON_SetState()</code>	Set the button state (handled automatically by touch module).
<code>BUTTON_SetStreamedBitmap()</code>	Set the bitmap used when displaying the button.
<code>BUTTON_SetText()</code>	Set the text.
<code>BUTTON_SetTextColor()</code>	Set the color(s) for the text.

## BUTTON\_Create()

### Description

Creates a BUTTON widget of a specified size at a specified location.

### Prototype

```
BUTTON_Handle BUTTON_Create(int x0, int y0, int xsize, int ysize,
```

```
int ID, int Flags);
```

Parameter	Meaning
<code>x0</code>	Leftmost pixel of the button (in desktop coordinates).
<code>y0</code>	Topmost pixel of the button (in desktop coordinates).
<code>xsize</code>	Horizontal size of the button (in pixels).
<code>ysize</code>	Vertical size of the button (in pixels).
<code>ID</code>	ID to be returned when button is pressed.
<code>Flags</code>	Window create flags. Typically <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 12: "The Window Manager" for a list of available parameter values).

### Return value

Handle for the created BUTTON widget; 0 if the routine fails.

## BUTTON\_CreateAsChild()

### Description

Creates a BUTTON widget as a child window.

### Prototype

```
BUTTON_Handle BUTTON_CreateAsChild(int x0, int y0, int xsize, int ysize,
                                   WM_HWIN hParent, int ID, int Flags);
```

Parameter	Meaning
<code>x0</code>	Leftmost pixel of the button (in desktop coordinates).
<code>y0</code>	Topmost pixel of the button (in desktop coordinates).
<code>xsize</code>	Horizontal size of the button (in pixels).
<code>ysize</code>	Vertical size of the button (in pixels).
<code>hParent</code>	Handle of parent window. If 0, the new button window will be a child of the desktop (top-level window).
<code>ID</code>	ID to be returned when button is pressed.
<code>Flags</code>	Window create flags (see <code>BUTTON_Create()</code> ).

### Return value

Handle for the created BUTTON widget; 0 if the routine fails.

## BUTTON\_CreateIndirect()

Prototype explained at the beginning of the chapter. The elements `Flags` and `Para` of the resource passed as parameter are not used.

## BUTTON\_SetBitmap()

### Description

Sets the bitmap(s) to be used when displaying a specified button.

### Prototype

```
void BUTTON_SetBitmap(BUTTON_Handle hObj, int Index,
```

```
const GUI_BITMAP* pBitmap);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of button.
<a href="#">Index</a>	Index for bitmap (see table below).
<a href="#">pBitmap</a>	Pointer to the bitmap structure.

Permitted values for parameter <a href="#">Index</a>	
0	Sets the bitmap to be used when button is unpressed.
1	Sets the bitmap to be used when button is pressed.

## BUTTON\_SetBitmapEx()

### Description

Sets the bitmap(s) to be used when displaying a specified button.

### Prototype

```
void BUTTON_SetBitmapEx(BUTTON_Handle hObj, int Index,
                        const GUI_BITMAP* pBitmap, int x, int y);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of button.
<a href="#">Index</a>	Index for bitmap (see <code>BUTTON_SetBitmap()</code> ).
<a href="#">pBitmap</a>	Pointer to the bitmap structure.
<a href="#">x</a>	X-position for the bitmap relative to the button.
<a href="#">y</a>	Y-position for the bitmap relative to the button.

## BUTTON\_SetBkColor()

### Description

Sets the button background color.

### Prototype

```
void BUTTON_SetBkColor(BUTTON_Handle hObj, int Index, GUI_COLOR Color);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of button.
<a href="#">Index</a>	Index for color (see table below).
<a href="#">Color</a>	Background color to be set.

Permitted values for parameter <a href="#">Index</a>	
0	Sets the color to be used when button is unpressed.
1	Sets the color to be used when button is pressed.

## BUTTON\_SetFont()

### Description

Sets the button font.

## Prototype

```
void BUTTON_SetFont(BUTTON_Handle hObj, const GUI_FONT* pFont);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of button.
<a href="#">pFont</a>	Pointer to the font.

## Additional information

If no font is selected, `BUTTON_FONT_DEF` will be used.

## BUTTON\_SetState()

### Description

Sets the state of a specified button object.

### Prototype

```
void BUTTON_SetState(BUTTON_Handle hObj, int State)
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of button.
<a href="#">State</a>	State of button (see table below).

Permitted values for parameter <a href="#">State</a>	
<code>BUTTON_STATE_PRESSED</code>	Button is pressed.
<code>BUTTON_STATE_INACTIVE</code>	Button is inactive.

## Additional information

This routine is used by the touch-panel module. You will not normally have to call this function.

## BUTTON\_SetStreamedBitmap()

### Description

Sets the streamed bitmap(s) to be used when displaying a specified button object.

### Prototype

```
void BUTTON_SetStreamedBitmap(BUTTON_Handle hObj, int Index,
                             const GUI_BITMAP_STREAM* pBitmap);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of button.
<a href="#">Index</a>	Index for bitmap (see <code>BUTTON_SetBitmap()</code> ).
<a href="#">pBitmap</a>	Pointer to a bitmap stream.

## Additional information

To be able to use this function you must include the following line in `LCDConf.h`:  

```
#define BUTTON_SUPPORT_STREAMED_BITMAP 1
```

For details about streamed bitmaps, please see `GUI_DrawStreamedBitmap()`.

**BUTTON\_SetText()**

**Description**  
Sets the text to be displayed on the button.

**Prototype**  
void BUTTON\_SetText(BUTTON\_Handle hObj, const char\* s);

Parameter	Meaning
hObj	Handle of button.
s	Text to display.

**BUTTON\_SetTextColor()**

**Description**  
Sets the button text color.

**Prototype**  
void BUTTON\_SetTextColor(BUTTON\_Handle hObj, int Index, GUI\_COLOR Color);

Parameter	Meaning
hObj	Handle of button.
Index	Index for text color (see BUTTON_SetBkColor( )).
Color	Text color to be set.

**Examples**

**Using the BUTTON widget**

The following example demonstrates how to use two bitmaps to create a simple button. It is available as WIDGET\_SimpleButton.c in the samples shipped with µC/GUI:

```

/*****
*                               Micrium Inc.                               *
*                               Empowering embedded systems               *
*                               µC/GUI sample code                       *
*                               ****                                     ****
*****/

-----
File      : WIDGET_SimpleButton.c
Purpose   : Example demonstrating the use of a BUTTON widget
-----
*/

#include "gui.h"
#include "button.h"

/*****
*                               ****                                     ****
*                               Demonstrates the use of a BUTTON widget   *
*                               ****                                     ****
*****/

static void DemoButton(void) {
    BUTTON_Handle hButton;
    int Key = 0;
    GUI_Init();
}
```





```

GUI_Init();
GUI_SetFont(&GUI_Font8x16);
GUI_DispStringHCenterAt("Click on phone button...", 160,0);
/* Create the button */
hButton = BUTTON_Create(142, 20, 36, 40, GUI_ID_OK, WM_CF_SHOW);
/* Modify the button attributes */
BUTTON_SetBkColor(hButton, 1, GUI_RED);
BUTTON_SetBitmapEx(hButton, 0, &bm_lbpp_0, 2, 4);
BUTTON_SetBitmapEx(hButton, 1, &bm_lbpp_1, 2, 4);
/* Loop until button is pressed */
while(GUI_GetKey() != GUI_ID_OK) {
    if (Stat ^= 1) {
        BUTTON_SetState(hButton,
                        BUTTON_STATE_HASFOCUS | BUTTON_STATE_INACTIVE);
    } else {
        BUTTON_SetState(hButton,
                        BUTTON_STATE_HASFOCUS | BUTTON_STATE_PRESSED);
    }
    GUI_Delay(500);
}
/* Delete button object */
BUTTON_Delete(hButton);
}

/*****
*
*           main
*
*****/

void main(void) {
    GUI_Init();
    DemoButton();
}





```

**Screen shot of above example**



## 13.4 CHECKBOX: Check box widget

One of the most familiar widgets for selecting various choices is the check box. A check box may be checked or unchecked by the user, and any number of boxes may be checked at one time. A box will appear gray if it is disabled, as seen in the table below where each of the four possible check box appearances are illustrated:

	Checked	Unchecked
<b>Enabled</b>		
<b>Disabled</b>		

All CHECKBOX-related routines are located in the file(s) `CHECKBOX*.c`, `CHECKBOX.h`. All identifiers are prefixed `CHECKBOX`.

## Configuration options

Type	Macro	Default	Explanation
N	<code>CHECKBOX_BKCOLOR0_DEFAULT</code>	0x808080	Background color, disabled state.
N	<code>CHECKBOX_BKCOLOR1_DEFAULT</code>	<code>GUI_WHITE</code>	Background color, enabled state.
N	<code>CHECKBOX_FGCOLOR0_DEFAULT</code>	0x101010	Foreground color, disabled state.
N	<code>CHECKBOX_FGCOLOR1_DEFAULT</code>	<code>GUI_BLACK</code>	Foreground color, enabled state.
S	<code>CHECKBOX_FONT_DEFAULT</code>	<code>&amp;GUI_Font13_1</code>	Font used for check mark.
B	<code>CHECKBOX_USE_3D</code>	1	Enable support for 3D effects.

## CHECKBOX API

The table below lists the available  $\mu$ C/GUI CHECKBOX-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<a href="#"><code>CHECKBOX_Check()</code></a>	Set the check box state to checked.
<a href="#"><code>CHECKBOX_Create()</code></a>	Create the check box.
<a href="#"><code>CHECKBOX_CreateIndirect()</code></a>	Create the check box from resource table entry.
<a href="#"><code>CHECKBOX_IsChecked()</code></a>	Return the current state (checked or not checked) of the check box.
<a href="#"><code>CHECKBOX_Uncheck()</code></a>	Set the check box state to unchecked (default).

## CHECKBOX\_Check()

### Description

Sets a specified check box to a checked state.

### Prototype

```
void CHECKBOX_Check(CHECKBOX_Handle hObj);
```

Parameter	Meaning
<a href="#"><code>hObj</code></a>	Handle of check box.

## CHECKBOX\_Create()

### Description

Creates a CHECKBOX widget of a specified size at a specified location.

### Prototype

```
CHECKBOX_Handle CHECKBOX_Create(int x0, int y0, int xsize, int ysize,
                                WM_HWIN hParent, int ID, int Flags);
```

Parameter	Meaning
<a href="#"><code>x0</code></a>	Leftmost pixel of the check box (in desktop coordinates).
<a href="#"><code>y0</code></a>	Topmost pixel of the check box (in desktop coordinates).
<a href="#"><code>xsize</code></a>	Horizontal size of the check box (in pixels).
<a href="#"><code>ysize</code></a>	Vertical size of the check box (in pixels).

Parameter	Meaning
<code>hParent</code>	Handle of parent window.
<code>ID</code>	ID to be returned.
<code>Flags</code>	Window create flags. Typically <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 12: "The Window Manager" for a list of available parameter values).

### Return value

Handle for the created CHECKBOX widget; 0 if the routine fails.

## CHECKBOX\_CreateIndirect()

Prototype explained at the beginning of the chapter. The elements `Flags` and `Para` of the resource passed as parameter are not used.

## CHECKBOX\_IsChecked()

### Description

Returns the current state (checked or not checked) of a specified CHECKBOX widget.

### Prototype

```
int CHECKBOX_IsChecked(CHECKBOX_Handle hObj);
```

Parameter	Meaning
<code>hObj</code>	Handle of check box.

### Return value

0: not checked

1: checked

## CHECKBOX\_Uncheck()

### Description

Sets a specified check box to an unchecked state.

### Prototype

```
void CHECKBOX_Uncheck(CHECKBOX_Handle hObj);
```

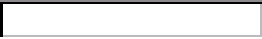

Parameter	Meaning
<code>hObj</code>	Handle of check box.

### Additional information



This is the default setting for check boxes.

## 13.5 EDIT: Edit widget

Edit fields are commonly used as the primary user interface for text input:

Blank edit field	Edit field with user input
	

You can also use edit fields for entering values in binary, decimal, or hexadecimal modes. A decimal-mode edit field might appear similar to those in the following table. Like a check box, an edit field will appear gray if disabled:

Edit field with user input (decimal)	Disabled edit field
	

All EDIT-related routines are located in the file(s) `EDIT*.c`, `EDIT.h`. All identifiers are prefixed EDIT.

### Configuration options

Type	Macro	Default	Explanation
S	EDIT_ALIGN_DEFAULT	GUI_TA_RIGHT   GUI_TA_VCENTER	Alignment for edit field text.
N	EDIT_BKCOLOR0_DEFAULT	0xc0c0c0	Background color, disabled state.
N	EDIT_BKCOLOR1_DEFAULT	GUI_WHITE	Background color, enabled state.
N	EDIT_BORDER_DEFAULT	1	Width of border, in pixels.
S	EDIT_FONT_DEFAULT	&GUI_Font13_1	Font used for edit field text.
N	EDIT_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color, disabled state.
N	EDIT_TEXTCOLOR1_DEFAULT	GUI_BLACK	Text color, enabled state.
N	EDIT_XOFF	2	Distance in X to offset text from left border of edit field.

Available alignment flags are:

GUI\_TA\_LEFT, GUI\_TA\_RIGHT, GUI\_TA\_HCENTER for horizontal alignment.

GUI\_TA\_TOP, GUI\_TA\_BOTTOM, GUI\_TA\_VCENTER for vertical alignment.

### EDIT API

The table below lists the available  $\mu$ C/GUI EDIT-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<a href="#">EDIT_AddKey()</a>	Key input routine.
<a href="#">EDIT_Create()</a>	Create the edit field.
<a href="#">EDIT_CreateIndirect()</a>	Create the edit field from resource table entry.
<a href="#">EDIT_GetDefaultFont()</a>	Returns the default font

Routine	Explanation
<code>EDIT_GetText()</code>	Get user input.
<code>EDIT_GetValue()</code>	Returns the current value
<code>EDIT_SetBinMode()</code>	Enables the binary edit mode
<code>EDIT_SetBkColor()</code>	Set the background color of the edit field.
<code>EDIT_SetDecMode()</code>	Enables the decimal edit mode
<code>EDIT_SetDefaultFont()</code>	Sets the default font used for edit fields.
<code>EDIT_SetDefaultTextAlign()</code>	Sets the default text alignment for edit fields.
<code>EDIT_SetFont()</code>	Select the font for the text.
<code>EDIT_SetHexMode()</code>	Enables the hexadecimal edit mode
<code>EDIT_SetMaxLen()</code>	Sets the maximum number of characters of the edit field.
<code>EDIT_SetText()</code>	Set the text.
<code>EDIT_SetTextAlign()</code>	Sets the text alignment for the edit field.
<code>EDIT_SetTextColor()</code>	Set the color(s) for the text.
<code>EDIT_SetValue()</code>	Sets the current value
<code>GUI_EditBin()</code>	Edits a binary value at the current cursor position
<code>GUI_EditDec()</code>	Edits a decimal value at the current cursor position
<code>GUI_EditHex()</code>	Edits a hexadecimal value at the current cursor position
<code>GUI_EditString()</code>	Edits a string at the current cursor position

## EDIT\_AddKey()

### Description

Adds user input to a specified edit field.

### Prototype

```
void EDIT_AddKey(EDIT_Handle hObj, int Key);
```

Parameter	Meaning
<code>hObj</code>	Handle of edit field.
<code>Key</code>	Character to be added.

### Additional information

The specified character is added to the user input of the EDIT widget. If you need to erase the last character you must call `GUI_ID_BACKSPACE`. If the maximum count of characters has been reached, another character will not be added.

## EDIT\_Create()

### Description

Creates an EDIT widget of a specified size at a specified location.

### Prototype

```
EDIT_Handle EDIT_Create(int x0, int y0, int xsize, int ysize,
```

```
int ID, int MaxLen, int Flags);
```

Parameter	Meaning
x0	Leftmost pixel of the edit field (in desktop coordinates).
y0	Topmost pixel of the edit field (in desktop coordinates).
xsize	Horizontal size of the edit field (in pixels).
ysize	Vertical size of the edit field (in pixels).
ID	ID to be returned.
MaxLen	Maximum count of characters.
Flags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow( ) in Chapter 12: "The Window Manager" for a list of available parameter values).

**Return value**

Handle for the created EDIT widget; 0 if the routine fails.

**EDIT\_CreateIndirect()**

Prototype explained at the beginning of the chapter. The element `Flags` of the resource passed as parameter is not used, but the maximum number of characters in the edit field is specified as the `Para` element.

**EDIT\_GetDefaultFont()**

**Description**

Sets the default font used for EDIT widgets.

**Prototype**

```
const GUI_FONT* EDIT_GetDefaultFont(void);
```

**Return value**

Default font used for EDIT widgets.

**EDIT\_GetText()**

**Description**

Retrieves the user input of a specified edit field.

**Prototype**

```
void EDIT_GetText(EDIT_Handle hObj, char* sDest, int MaxLen);
```

Parameter	Meaning
hObj	Handle of edit field.
sDest	Pointer to buffer.
MaxLen	Size of buffer.

**EDIT\_GetValue()**

**Description**

Returns the current value of the edit field. The current value is only useful if the edit field is in binary, decimal or hexadecimal mode.

## Prototype

```
I32 EDIT_GetValue(EDIT_Handle hObj);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of edit field.

## Return value

The current value.

## EDIT\_SetBinMode()

### Description

Enables the binary edit mode of the edit field. The given value can be modified in the given range.

### Prototype

```
void EDIT_SetBinMode(EDIT_Handle hObj, U32 Value, U32 Min, U32 Max);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of edit field.
<a href="#">Value</a>	Value to be modified.
<a href="#">Min</a>	Minimum value.
<a href="#">Max</a>	Maximum value.

## EDIT\_SetBkColor()

### Description

Sets the edit field background color.

### Prototype

```
void EDIT_SetBkColor(EDIT_Handle hObj, int Index, GUI_COLOR Color);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of edit field.
<a href="#">Index</a>	Has to be 0, reserved for future use.
<a href="#">Color</a>	Color to be set.

## EDIT\_SetDecMode()

### Description

Enables the decimal edit mode of the edit field. The given value can be modified in the given range.

### Prototype

```
void EDIT_SetDecMode(EDIT_Handle hEdit, I32 Value, I32 Min, I32 Max,
```

```
int Shift, U8 Flags);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of edit field.
<a href="#">Value</a>	Value to be modified.
<a href="#">Min</a>	Minimum value.
<a href="#">Max</a>	Maximum value.
<a href="#">Shift</a>	If > 0 it specifies the position of the decimal point.
<a href="#">Flags</a>	See table below.

Permitted values for parameter <a href="#">Flags</a> ("OR" combinable)	
0 (default)	Edit in normal mode, sign is only displayed if the value is negative
GUI_EDIT_SIGNED	"+" and "-" sign is displayed.

## EDIT\_SetDefaultFont()

### Description

Sets the default font used for edit fields.

### Prototype

```
void EDIT_SetDefaultFont(const GUI_FONT* pFont);
```

Parameter	Meaning
<a href="#">pFont</a>	Pointer to the font to be set as default.

## EDIT\_SetDefaultTextAlign()

### Description

Sets the default text alignment for edit fields.

### Prototype

```
void EDIT_SetDefaultTextAlign(int Align);
```

Parameter	Meaning
<a href="#">Align</a>	Default text alignment (see <a href="#">GUI_SetTextAlign()</a> )

## EDIT\_SetFont()

### Description

Sets the edit field font.

### Prototype

```
void EDIT_SetFont(EDIT_Handle hObj, const GUI_FONT* pfont);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of edit field.
<a href="#">pFont</a>	Pointer to the font.

## EDIT\_SetHexMode()

### Description

Enables the hexadecimal edit mode of the edit field. The given value can be modified in the given range.

### Prototype

```
void EDIT_SetHexMode(EDIT_Handle hObj, U32 Value, U32 Min, U32 Max);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of edit field.
<a href="#">Value</a>	Value to be modified.
<a href="#">Min</a>	Minimum value.
<a href="#">Max</a>	Maximum value.

## EDIT\_SetMaxLen()

### Description

Sets the maximum number of characters to be edited by the given edit field.

### Prototype

```
void EDIT_SetMaxLen(EDIT_Handle hObj, int MaxLen);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of edit field.
<a href="#">MaxLen</a>	Number of characters.

## EDIT\_SetText()

### Description

Sets the text to be displayed in the edit field.

### Prototype

```
void EDIT_SetText(EDIT_Handle hObj, const char* s)
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of edit field.
<a href="#">s</a>	Text to display.

## EDIT\_SetTextAlign()

### Description

Sets the text alignment of the edit field.

### Prototype

```
void EDIT_SetTextAlign(EDIT_Handle hObj, int Align);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of edit field.
<a href="#">Align</a>	Text alignment to be set (see <a href="#">GUI_SetTextAlign()</a> )

## EDIT\_SetTextColor()

### Description

Sets the edit field text color.

### Prototype

```
void EDIT_SetBkColor(EDIT_Handle hObj, int Index, GUI_COLOR Color);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of edit field.
<a href="#">Index</a>	Has to be 0, reserved for future use.
<a href="#">Color</a>	Color to be set.

## EDIT\_SetValue()

### Description

Sets the current value of the edit field. Only useful if binary, decimal or hexadecimal edit mode is set.

### Prototype

```
void EDIT_SetValue(EDIT_Handle hObj, I32 Value);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of edit field.
<a href="#">Value</a>	New value

## GUI\_EditBin()

### Description

Edits a binary value at the current cursor position.

### Prototype

```
U32 GUI_EditBin(U32 Value, U32 Min, U32 Max, int Len, int xsize);
```

Parameter	Meaning
<a href="#">Value</a>	Value to be modified.
<a href="#">Min</a>	Minimum value.
<a href="#">Max</a>	Maximum value.
<a href="#">Len</a>	Number of digits to edit.
<a href="#">xsize</a>	Pixel-size in X of the edit field.

### Return value

The new value will be returned if <ENTER> is pressed. If <ESC> is pressed, the old value is returned.

### Additional information

The routine returns after pressing <ENTER> or <ESC>. The contents of the given text will be modified only if <ENTER> is pressed.

## GUI\_EditDec()

### Description

Edits a decimal value at the current cursor position.

### Prototype

```
U32 GUI_EditDec(I32 Value, I32 Min, I32 Max, int Len, int xsize,
               int Shift, U8 Flags);
```

Parameter	Meaning
<a href="#">Value</a>	Value to be modified.
<a href="#">Min</a>	Minimum value.
<a href="#">Max</a>	Maximum value.
<a href="#">Len</a>	Number of digits to edit.
<a href="#">xsize</a>	Pixel-size in X of edit field.
<a href="#">Shift</a>	If > 0 it specifies the position of the decimal point.
<a href="#">Flags</a>	See <code>EDIT_SetDecMode()</code> .

### Return value

The new value will be returned if <ENTER> is pressed. If <ESC> is pressed, the old value is returned.

### Additional information

The routine returns after pressing <ENTER> or <ESC>. The contents of the given text will be modified only if <ENTER> is pressed.

## GUI\_EditHex()

### Description

Edits a hexadecimal value at the current cursor position.

### Prototype

```
U32 GUI_EditHex(U32 Value, U32 Min, U32 Max, int Len, int xsize);
```

Parameter	Meaning
<a href="#">Value</a>	Value to be modified.
<a href="#">Min</a>	Minimum value.
<a href="#">Max</a>	Maximum value.
<a href="#">Len</a>	Number of digits to edit.
<a href="#">xsize</a>	Pixel-size in X of the edit field.

### Return value

The new value will be returned if <ENTER> is pressed. If <ESC> is pressed, the old value is returned.

### Additional information

The routine returns after pressing <ENTER> or <ESC>. The contents of the given text will be modified only if <ENTER> is pressed.

## GUI\_EditString()

### Description

Edits a string at the current cursor position.

### Prototype

```
void GUI_EditString(char * pString, int Len, int xsize);
```

Parameter	Meaning
<code>pString</code>	Pointer to the string to be edited.
<code>Len</code>	Maximum number of characters.
<code>xsize</code>	Pixel-size in X of the edit field.

### Additional information

The routine returns after pressing <ENTER> or <ESC>. The contents of the given text will be modified only if <ENTER> is pressed.

## Example

The following example demonstrates the use of the EDIT widget. It is available as `WIDGET_Edit.c` in the samples shipped with `µC/GUI`:

```

/*****
 *                               *
 *           Micrium Inc.       *
 *      Empowering embedded systems      *
 *                               *
 *           µC/GUI sample code      *
 *                               *
 *****/

-----
File       : WIDGET_Edit.c
Purpose    : Example demonstrating the use of a EDIT widget
-----
*/

#include "gui.H"
#include "edit.h"

/*****
 *                               *
 *      Edit a string until ESC or ENTER is pressed      *
 *                               *
 *****/

static int Edit(void) {
    int Key;
    EDIT_Handle hEdit;
    char aBuffer[28];
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringHCenterAt("Use keyboard to modify string...", 160, 0);
    /* Create edit widget */
    hEdit = EDIT_Create( 50, 20, 219, 25, ' ', sizeof(aBuffer), 0 );
    /* Modify edit widget */
    EDIT_SetText(hEdit, "Press <ENTER> when done...");
    EDIT_SetFont(hEdit, &GUI_Font8x16);
    EDIT_SetTextColor(hEdit, 0, GUI_RED);
    /* Handle keyboard until ESC or ENTER is pressed */
    do {
        Key = GUI_WaitKey();
        switch (Key) {
            case GUI_ID_ESCAPE:
            case GUI_ID_CANCEL:

```

```

        break;
    default:
        EDIT_AddKey(hEdit, Key);
    }
} while ((Key != GUI_ID_ESCAPE) && (Key != GUI_ID_ENTER) && (Key != 0));
/* Fetch result from edit widget */
if (Key == GUI_ID_ENTER)
    EDIT_GetText(hEdit, aBuffer, sizeof(aBuffer));
else
    aBuffer[0] = 0;
    EDIT_Delete(hEdit);
    GUI_DispatchStringHCenterAt(aBuffer, 160, 50);
    return Key;
}

/*****
*
*                               main
*
*****/

void main(void) {
    GUI_Init();
    Edit();
    while(1)
        GUI_Delay(100);
}



```

**Screen shot of above example**



## 13.6 FRAMEWIN: Frame window widget

Frame windows give your application a PC application-window appearance. They consist of a surrounding frame, a title bar and a user area. The color of the title bar changes to show whether the window is active or inactive, as seen below:

Active frame window	Inactive frame window
	

All FRAMEWIN-related routines are located in the file(s) FRAMEWIN\*.c, FRAMEWIN.h. All identifiers are prefixed FRAMEWIN.

## Configuration options

Type	Macro	Default	Explanation
N	FRAMEWIN_BARCOLOR_ACTIVE_DEFAULT	0xff0000	Title bar color, active state.
N	FRAMEWIN_BARCOLOR_INACTIVE_DEFAULT	0x404040	Title bar color, inactive state.
N	FRAMEWIN_BORDER_DEFAULT	3	Outer border width, in pixels.
N	FRAMEWIN_CLIENTCOLOR_DEFAULT	0xc0c0c0	Color of client window area.
S	FRAMEWIN_DEFAULT_FONT	&GUI_Font8_1	Font used for title bar text.
N	FRAMEWIN_FRAMECOLOR_DEFAULT	0xaaaaaa	Frame color.
N	FRAMEWIN_IBORDER_DEFAULT	1	Inner border width, in pixels.
B	FRAMEWIN_USE_3D	1	Enable support for 3D effects.

## FRAMEWIN API

The table below lists the available  $\mu$ C/GUI FRAMEWIN-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<a href="#">FRAMEWIN_Create()</a>	Creates the frame window.
<a href="#">FRAMEWIN_CreateAsChild()</a>	Create the frame window as a child window.
<a href="#">FRAMEWIN_CreateIndirect()</a>	Create the frame window from resource table entry.
<a href="#">FRAMEWIN_GetDefaultBorderSize()</a>	Returns the default border size
<a href="#">FRAMEWIN_GetDefaultCaptionSize()</a>	Returns the default size of the title bar
<a href="#">FRAMEWIN_GetDefaultFont()</a>	Returns the default font used for the title bar
<a href="#">FRAMEWIN_SetActive()</a>	Sets the state of the frame window
<a href="#">FRAMEWIN_SetBarColor()</a>	Sets the background color of the title bar.
<a href="#">FRAMEWIN_SetClientColor()</a>	Sets the background color of the client area
<a href="#">FRAMEWIN_SetDefaultBarColor()</a>	Sets the default color of the title bar
<a href="#">FRAMEWIN_SetDefaultBorderSize()</a>	Sets the default border size
<a href="#">FRAMEWIN_SetDefaultCaptionSize()</a>	Sets the default height of the title bar
<a href="#">FRAMEWIN_SetDefaultFont()</a>	Sets the default font of the title bar.
<a href="#">FRAMEWIN_SetFont()</a>	Selects the font for the title text.
<a href="#">FRAMEWIN_SetText()</a>	Sets the title text.
<a href="#">FRAMEWIN_SetTextAlign()</a>	Sets the alignment of the title text.
<a href="#">FRAMEWIN_SetTextColor()</a>	Sets the color(s) for the title text.
<a href="#">FRAMEWIN_SetTextPos()</a>	Sets the position of the title text.

## FRAMEWIN\_Create()

### Description

Creates a FRAMEWIN widget of a specified size at a specified location.

### Prototype

```
FRAMEWIN_Handle FRAMEWIN_Create(const char* pTitle, WM_CALLBACK* cb,
                                int Flags, int x0, int y0,
```

```
int xsize, int ysize);
```

Parameter	Meaning
<code>pTitle</code>	Title displayed in the title bar.
<code>cb</code>	(Reserved for future use).
<code>Flags</code>	Window create flags. Typically <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 12: "The Window Manager" for a list of available parameter values).
<code>x0</code>	X-position of the frame window.
<code>y0</code>	Y-position of the frame window.
<code>xsize</code>	X-size of the framw window.
<code>ysize</code>	Y-size of the frame window.

### Return value

Handle for the created FRAMEWIN widget; 0 if the routine fails.

## FRAMEWIN\_CreateAsChild()

### Description

Creates a FRAMEWIN widget as a child window.

### Prototype

```
FRAMEWIN_Handle FRAMEWIN_CreateAsChild(int x0, int y0, int xsize, int ysize,
                                       WM_HWIN hParent, const char* pText,
                                       WM_CALLBACK* cb, int Flags);
```

Parameter	Meaning
<code>x0</code>	X-position of the frame window relative to the parent window.
<code>y0</code>	Y-position of the frame window relative to the parent window.
<code>xsize</code>	X-size of the frame window.
<code>ysize</code>	Y-size of the frame window.
<code>hParent</code>	Handle of parent window.
<code>pText</code>	Text to be displayed in the title bar.
<code>cb</code>	(Reserved for future use).
<code>Flags</code>	Window create flags (see <code>FRAMEWIN_Create()</code> ).

### Return value

Handle for the created FRAMEWIN widget; 0 if the routine fails.

## FRAMEWIN\_CreateIndirect()

Prototype explained at the beginning of the chapter. The following flag may be used as the `Flags` element of the resource passed as parameter:

Permitted indirect creation flag	
<code>FRAMEWIN_CF_MOVEABLE</code>	Makes the frame window moveable (default is fixed).

The `Para` element is not used in the resource table.

## FRAMEWIN\_GetDefaultBorderSize()

**Description**  
Returns the default size of a frame window border.

**Prototype**  
`int FRAMEWIN_GetDefaultBorderSize(void);`

**Return value**  
Default size for a frame window border.

## FRAMEWIN\_GetDefaultCaptionSize()

**Description**  
Returns the default height of the title bar of frame windows.

**Prototype**  
`int FRAMEWIN_GetDefaultCaptionSize(void);`

**Return value**  
Default height of the title bar.

## FRAMEWIN\_GetDefaultFont()

**Description**  
Returns the default font used for frame window captions.

**Prototype**  
`const GUI_FONT* FRAMEWIN_GetDefaultFont(void);`

**Return value**  
Default font used for frame window captions.

## FRAMEWIN\_SetActive()

**Description**  
Sets the state of frame window. Depending on the state, the color of the title bar will change.

**Prototype**  
`void FRAMEWIN_SetActive(FRAMEWIN_Handle hObj, int State);`

Parameter	Meaning
<code>hObj</code>	Handle of frame window.
<code>State</code>	State of frame window (see table below).

Permitted values for parameter <code>State</code>	
0	Frame window is inactive.
1	Frame window is active.

## FRAMEWIN\_SetBarColor()

### Description

Sets the background color of the title bar.

### Prototype

```
void FRAMEWIN_SetBarColor(FRAMEWIN_Handle hObj, int Index, GUI_COLOR Color);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of frame window.
<a href="#">Index</a>	Has to be 0; reserved for future use.
<a href="#">Color</a>	Color to be used as background color for the title bar.

## FRAMEWIN\_SetClientColor()

### Description

Sets the color of the client window area.

### Prototype

```
void FRAMEWIN_SetClientColor(FRAMEWIN_Handle hObj, GUI_COLOR Color);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of frame window.
<a href="#">Color</a>	Color to be set.

## FRAMEWIN\_SetDefaultBarColor()

### Description

Sets the color of the title bar.

### Prototype

```
void FRAMEWIN_SetDefaultBarColor(int Index, GUI_COLOR Color);
```

Parameter	Meaning
<a href="#">Index</a>	Index for color (see table below).
<a href="#">Color</a>	Color to be set.

Permitted values for parameter <a href="#">Index</a>	
0	Sets the color to be used when frame window is inactive.
1	Sets the color to be used when frame window is active.

## FRAMEWIN\_SetDefaultBorderSize()

### Description

Sets the default border size of frame windows.

## Prototype

```
void FRAMEWIN_SetDefaultBorderSize(int BorderSize);
```

Parameter	Meaning
<a href="#">BorderSize</a>	Size to be set.

## FRAMEWIN\_SetDefaultCaptionSize()

### Description

Sets the size in Y for the title bar.

### Prototype

```
void FRAMEWIN_SetDefaultCaptionSize(int CaptionSize);
```

Parameter	Meaning
<a href="#">CaptionSize</a>	Size to be set

## FRAMEWIN\_SetDefaultFont()

### Description

Sets the default font used to display the title.

### Prototype

```
void FRAMEWIN_SetDefaultFont(const GUI_FONT* pFont);
```

Parameter	Meaning
<a href="#">pFont</a>	Pointer to font to be used as default

## FRAMEWIN\_SetFont()

### Description

Sets the title font.

### Prototype

```
void FRAMEWIN_SetFont(FRAMEWIN_Handle hObj, const GUI_FONT* pfont);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of frame window.
<a href="#">pFont</a>	Pointer to the font.

## FRAMEWIN\_SetText()

### Description

Sets the title text.

## Prototype

```
void FRAMEWIN_SetText(FRAMEWIN_Handle hObj, const char* s);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of frame window.
<a href="#">s</a>	Text to display as the title.

## FRAMEWIN\_SetTextAlign()

### Description

Sets the text alignment of the title bar.

### Prototype

```
void FRAMEWIN_SetTextAlign(FRAMEWIN_Handle hObj, int Align);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of frame window.
<a href="#">Align</a>	Alignment attribute for the title (see table below).

Permitted values for parameter <a href="#">Align</a>	
GUI_TA_HCENTER	Centers the title (default).
GUI_TA_LEFT	Displays the title to the left.
GUI_TA_RIGHT	Displays the title to the right.

### Additional information

If this function is not called, the default behavior is to display the text centered.

## FRAMEWIN\_SetTextColor()

### Description

Sets the title text color.

### Prototype

```
void FRAMEWIN_SetTextColor(FRAMEWIN_Handle hObj, GUI_COLOR Color);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of frame window.
<a href="#">Color</a>	Color to be used for title text.

## FRAMEWIN\_SetTextPos()

### Description

Sets the position of the title text.

## Prototype

```
void FRAMEWIN_SetTextPos(FRAMEWIN_Handle hObj, int XOff, int YOff);
```

Parameter	Meaning
<code>hObj</code>	Handle of frame window.
<code>XOff</code>	X-position of the title text.
<code>YOff</code>	Y-position of the title text.

## Example

The following example illustrates how to use the FRAMEWIN widget. The widget is first created, several of its attributes are modified, and then a client window is created. You cannot use the frame window widget itself to display something; a client window must always be created. This example can be found in the samples as WIDGET\_FrameWin.c:

```

/*****
*                               Micrium Inc.                               *
*                               Empowering embedded systems                 *
*                               *                                           *
*                               µC/GUI sample code                         *
*                               *                                           *
*****/

-----
File      : WIDGET_FrameWin.c
Purpose   : Example demonstrating the use of a FRAMEWIN widget
-----
*/

#include "gui.H"
#include "framewin.h"
#include <stddef.h>

/*****
*                               *
*                               Callback routine of client window          *
*                               *
*****/

static WM_RESULT CallbackChild(WM_MESSAGE * pMsg) {
    WM_HWIN hWin = (FRAMEWIN_Handle)(pMsg->hWin);
    switch (pMsg->MsgId) {
        case WM_PAINT:
            /* Handle the paint message */
            GUI_SetBkColor(GUI_BLUE);
            GUI_SetColor(GUI_WHITE);
            GUI_SetFont(&GUI_FontComic24B_ASCII);
            GUI_SetTextAlign(GUI_TA_HCENTER | GUI_TA_VCENTER);
            GUI_Clear();
            GUI_DispStringHCenterAt("Child window",
                                    WM_GetWindowSizeX(hWin) / 2,
                                    WM_GetWindowSizeY(hWin) / 2);
            break;
    }
}

/*****
*                               *
*                               Create frame- and childwindow              *
*                               *
*****/

static void DemoFramewin(void) {

```

```

/* Create frame window */
FRAMEWIN_Handle hFrame = FRAMEWIN_Create("Frame window",
                                          NULL, WM_CF_SHOW,
                                          10, 10, 150, 60);

FRAMEWIN_SetFont(hFrame, &GUI_Font16B_ASCII);
FRAMEWIN_SetTextColor(hFrame, GUI_RED);
FRAMEWIN_SetBarColor(hFrame, 0, GUI_GREEN);
FRAMEWIN_SetTextAlign(hFrame, GUI_TA_HCENTER);
/* Create client window */
WM_CreateWindowAsChild(0, 0, 0, 0, hFrame, WM_CF_SHOW, CallbackChild, 0);
}

/*****
*
*           main
*
*****/

void main(void) {
    GUI_Init();
    DemoFrameWin();
    while(1)
        GUI_Delay(100);
}

```

**Screen shot of above example**



## 13.7 LISTBOX: List box widget

List boxes are used to select one element of a list. A list box can be created without a surrounding frame window, as shown below, or as a child window of a FRAMEWIN widget (see the additional screen shots at the end of the section). As items in a list box are selected, they appear highlighted. Note that the background color of a selected item depends on whether the list box window has input focus.

List box with focus	List box without focus

All LISTBOX-related routines are in the file(s) LISTBOX\*.c, LISTBOX.h. All identifiers are prefixed LISTBOX.

## Configuration options

Type	Macro	Default	Explanation
N	LISTBOX_BKCOLOR0_DEFAULT	GUI_WHITE	Background color, unselected state.
N	LISTBOX_BKCOLOR1_DEFAULT	GUI_GRAY	Background color, selected state without focus.
N	LISTBOX_BKCOLOR2_DEFAULT	GUI_WHITE	Background color, selected state with focus.
S	LISTBOX_FONT_DEFAULT	&GUI_Font13_1	Font used.
N	LISTBOX_TEXTCOLOR0_DEFAULT	GUI_BLACK	Text color, unselected state.
N	LISTBOX_TEXTCOLOR1_DEFAULT	GUI_BLACK	Text color, selected state without focus.
N	LISTBOX_TEXTCOLOR2_DEFAULT	GUI_BLACK	Text color, selected state with focus.
B	LISTBOX_USE_3D	1	Enable 3D support.

## LISTBOX API

The table below lists the available  $\mu$ C/GUI LISTBOX-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<a href="#">LISTBOX_Create()</a>	Create the list box.
<a href="#">LISTBOX_CreateAsChild()</a>	Create the list box as a child window.
<a href="#">LISTBOX_CreateIndirect()</a>	Create the list box from resource table entry.
<a href="#">LISTBOX_DecSel()</a>	Decrement selection.
<a href="#">LISTBOX_GetDefaultFont()</a>	Return the default font for LISTBOX widgets.
<a href="#">LISTBOX_GetSel()</a>	Return the number of the selected row.
<a href="#">LISTBOX_IncSel()</a>	Increment selection.
<a href="#">LISTBOX_SetBackColor()</a>	Set the background color.
<a href="#">LISTBOX_SetDefaultFont()</a>	Change the default font for LISTBOX widgets.
<a href="#">LISTBOX_SetFont()</a>	Select the font.
<a href="#">LISTBOX_SetSel()</a>	Set the selected row.
<a href="#">LISTBOX_SetTextColor()</a>	Set the foreground color.

## LISTBOX\_Create()

### Description

Creates a LISTBOX widget of a specified size at a specified location.

### Prototype

```
LISTBOX_Handle LISTBOX_Create(const GUI_ConstString* ppText, int x0, int y0,
                             int xSize, int ySize, int Flags);
```

Parameter	Meaning
<a href="#">ppText</a>	Pointer to an array of string pointers containing the elements to be displayed.
<a href="#">x0</a>	X-position of the list box.
<a href="#">y0</a>	Y-position of the list box.

Parameter	Meaning
<code>xSize</code>	X-size of the list box.
<code>ySize</code>	Y-size of the list box.
<code>Flags</code>	Window create flags. Typically <code>WM_CF_SHOW</code> in order to make the widget visible immediately (please refer to <code>WM_CreateWindow()</code> in Chapter 12: "The Window Manager" for a list of available parameter values).

### Return value

Handle for the created FRAMEWIN widget, 0 on failure.

### Additional information

If the parameter `ySize` is greater than the required space for drawing the contents of the widget, the Y-size will be reduced to the required value. The same applies for the `xSize` parameter.

## LISTBOX\_CreateAsChild()

### Description

Creates a LISTBOX widget as a child window.

### Prototype

```
LISTBOX_Handle LISTBOX_CreateAsChild(const GUI_ConstString* ppText,
                                     HBWIN hWinParent, int x0, int y0,
                                     int xSize, int ySize, int Flags);
```

Parameter	Meaning
<code>ppText</code>	Pointer to an array of string pointers containing the elements to be displayed.
<code>hWinParent</code>	Handle of parent window.
<code>x0</code>	X-position of the list box relative to the parent window.
<code>y0</code>	Y-position of the list box relative to the parent window.
<code>xSize</code>	X-size of the list box
<code>ySize</code>	Y-size of the list box.
<code>Flags</code>	Window create flags (see <code>LISTBOX_Create()</code> ).

### Return value

Handle for the created LISTBOX widget; 0 if the routine fails.

### Additional information

If the parameter `ySize` is greater than the space required for drawing the contents of the widget, the Y-size will be reduced to the required value. If `ySize` = 0 the Y-size of the widget will be set to the Y-size of the client area from the parent window. The same applies for the `xSize` parameter.

## LISTBOX\_CreateIndirect()

Prototype explained at the beginning of the chapter. The elements `Flags` and `Para` of the resource passed as parameter are not used.

## LISTBOX\_DecSel()

### Description

Decrement the list box selection (moves the selection bar of a specified list box up by one element).

### Prototypes

```
void LISTBOX_DecSel(LISTBOX_Handle hObj);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of list box.

## LISTBOX\_GetDefaultFont()

### Description

Returns the default font used for creating LISTBOX widgets.

### Prototype

```
const GUI_FONT* LISTBOX_GetDefaultFont(void);
```

### Return value

Pointer to default font.

## LISTBOX\_GetSel()

### Description

Returns the number of the currently selected element in a specified list box.

### Prototype

```
int LISTBOX_GetSel(LISTBOX_Handle hObj);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of list box.

### Return value

Number of the currently selected element.

## LISTBOX\_IncSel()

### Description

Increment the list box selection (moves the selection bar of a specified list box down by one element).

### Prototypes

```
void LISTBOX_IncSel(LISTBOX_Handle hObj);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of list box.

## LISTBOX\_SetBackColor()

### Description

Sets the list box background color.

### Prototype

```
void LISTBOX_SetBackColor(LISTBOX_Handle hObj, int Index, GUI_COLOR Color);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of list box.
<a href="#">Index</a>	Index for background color (see table below).
<a href="#">Color</a>	Color to be set.

Permitted values for parameter <a href="#">Index</a>	
0	Sets the color for unselected elements.
1	Sets the color for the selected element.

## LISTBOX\_SetDefaultFont()

### Description

Sets the default font used for creating LISTBOX widgets.

### Prototype

```
void LISTBOX_SetDefaultFont(const GUI_FONT* pFont)
```

Parameter	Meaning
<a href="#">pFont</a>	Pointer to the font.

## LISTBOX\_SetFont()

### Description

Sets the list box font.

### Prototype

```
void LISTBOX_SetFont(LISTBOX_Handle hObj, const GUI_FONT* pfont);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of list box.
<a href="#">pFont</a>	Pointer to the font.

## LISTBOX\_SetSel()

### Description

Sets the selected element of a specified list box.

Prototype

```
void LISTBOX_SetSel(LISTBOX_Handle hObj, int Sel);
```

Parameter	Meaning
<code>hObj</code>	Handle of list box.
<code>Sel</code>	Elenent to be selected.

LISTBOX\_SetTextColor()

Description

Sets the list box text color.

Prototype

```
void LISTBOX_SetTextColor(LISTBOX_Handle hObj, int Index, GUI_COLOR Color);
```

Parameter	Meaning
<code>hObj</code>	Handle of list box.
<code>Index</code>	Index for text color (see LISTBOX_SetBackColor()).
<code>Color</code>	Color to be set.

Examples

Using the LISTBOX widget

The following example demonstrates the simple use of the LISTBOX widget. It is available in the samples as WIDGET\_SimpleListBox.c:

```

/*****
*                               Micrium Inc.                               *
*                               Empowering embedded systems               *
*                               uC/GUI sample code                       *
*                               ****                                     ****
*****/

-----
File      : WIDGET_SimpleListBox.c
Purpose   : Example demonstrating the LISTBOX widget
-----
*/

#include "gui.H"
#include "listbox.h"
#include <stddef.h>

const GUI_ConstString ListBox[] = {
    "English", "Deutsch", "Français", "Japanese", "Italiano", NULL
};

#define countof(Array) (sizeof(Array) / sizeof(Array[0]))

/*****
*                               Simple use of the LISTBOX widget         *
*                               ****                                     ****
*****/

void DemoListBox(void) {
    int i;
    int Entries = countof(ListBox) - 1;
    LISTBOX_Handle hListBox;
```

```

int ySize = GUI_GetFontSizeYOf(&GUI_Fontl3B_1) * Entries;
/* Create the listbox */
hListBox = LISTBOX_Create(ListBox, 10, 10, 60, ySize, WM_CF_SHOW);
/* Change current selection of the listbox */
for (i = 0; i < Entries-1; i++) {
    GUI_Delay(500);
    LISTBOX_IncSel(hListBox);
    WM_Exec();
}
for (i = 0; i < Entries-1; i++) {
    GUI_Delay(500);
    LISTBOX_DecSel(hListBox);
    WM_Exec();
}
/* Delete listbox widget */
LISTBOX_Delete(hListBox);
GUI_Clear();
}

/*****
 *
 *          Demonstrates LISTBOX widget
 *
 *****/

static void DemoListbox(void) {
    GUI_SetBkColor(GUI_BLUE);
    GUI_Clear();
    while(1) {
        DemoListBox();
    }
}

/*****
 *
 *          main
 *
 *****/

void main(void) {
    GUI_Init();
    DemoListbox();
}

```

### Screen shot of above example



### Using the LISTBOX widget with and without frames

The following example illustrates the use of the LISTBOX widget with and without a frame window. It is available as `WIDGET_ListBox.c`:

```

/*****
 *
 *          Micrium Inc.
 *          Empowering embedded systems
 *
 *          µC/GUI sample code
 *
 *****/

-----
File       : WIDGET_ListBox.c
Purpose    : Example demonstrating the LISTBOX widget
-----

```

```

*/

#include "gui.H"
#include "framewin.h"
#include "listbox.h"
#include <stddef.h>

const GUI_ConstString ListBox[] = {
    "English", "Deutsch", "Français", "Japanese", "Italiano", NULL
};

#define countof(Array) (sizeof(Array) / sizeof(Array[0]))

/*****
 *
 *          Uses only the LISTBOX widget
 *
 *****/

void DemoListBox(void) {
    int i;
    int Entries = countof(ListBox) - 1;
    LISTBOX_Handle hListBox;
    int ySize = GUI_GetFontSizeYOf(&GUI_Fontl3B_1) * Entries;
    /* Create the listbox */
    hListBox = LISTBOX_Create(ListBox, 10, 10, 120, ySize, WM_CF_SHOW);
    /* Modify listbox attributes */
    LISTBOX_SetFont(hListBox, &GUI_Fontl3B_1);
    LISTBOX_SetBackColor(hListBox, 0, GUI_BLUE);
    LISTBOX_SetBackColor(hListBox, 1, GUI_LIGHTBLUE);
    LISTBOX_SetTextColor(hListBox, 0, GUI_WHITE);
    LISTBOX_SetTextColor(hListBox, 1, GUI_BLACK);
    /* Change current selection of the listbox */
    for (i = 0; i < Entries - 1; i++) {
        GUI_Delay(500);
        LISTBOX_IncSel(hListBox);
    }
    for (i = 0; i < Entries - 1; i++) {
        GUI_Delay(500);
        LISTBOX_DecSel(hListBox);
    }
    GUI_Delay(500);
    /* Delete listbox widget */
    LISTBOX_Delete(hListBox);
    GUI_Clear();
}

/*****
 *
 *          Uses LISTBOX widget as child of a FRAMEWIN widget
 *
 *****/

void DemoListBoxAsChild(void) {
    int i;
    int Entries = countof(ListBox) - 1;
    FRAMEWIN_Handle hFrame;
    LISTBOX_Handle hListBox;
    int ySize = GUI_GetFontSizeYOf(&GUI_Fontl3B_1) * Entries
        + GUI_GetFontSizeYOf(&GUI_Fontl6B_ASCII)
        + 7;
    /* Create the frame window */
    hFrame = FRAMEWIN_Create("List box", NULL, WM_CF_SHOW, 10, 10, 120, ySize);
    /* Modify frame window attributes */
    FRAMEWIN_SetFont(hFrame, &GUI_Fontl6B_ASCII);
    FRAMEWIN_SetTextColor(hFrame, GUI_RED);
    FRAMEWIN_SetBarColor(hFrame, 0, GUI_GREEN);
    FRAMEWIN_SetTextAlign(hFrame, GUI_TA_HCENTER);
    /* Create the listbox */
    hListBox = LISTBOX_CreateAsChild(ListBox, hFrame, 0, 0, 0, 0, WM_CF_SHOW);
    /* Modify listbox attributes */

```

```

LISTBOX_SetBackColor(hListBox, 0, GUI_BLUE);
LISTBOX_SetBackColor(hListBox, 1, GUI_LIGHTBLUE);
LISTBOX_SetTextColor(hListBox, 0, GUI_WHITE);
LISTBOX_SetTextColor(hListBox, 1, GUI_BLACK);
LISTBOX_SetFont(hListBox, &GUI_Font13B_1);
/* Change current selection of the listbox */
for (i = 0; i < Entries - 1; i++) {
    GUI_Delay(500);
    LISTBOX_IncSel(hListBox);
}
for (i = 0; i < Entries - 1; i++) {
    GUI_Delay(500);
    LISTBOX_DecSel(hListBox);
}
GUI_Delay(500);
/* Delete listbox widget */
LISTBOX_Delete(hListBox);
/* Delete framewin widget */
FRAMEWIN_Delete(hFrame);
GUI_Clear();
}

/*****
*
*           Demonstrates LISTBOX widget
*
*****/

static void DemoListbox(void) {
    GUI_SetBkColor(GUI_GRAY);
    GUI_Clear();
    while(1) {
        DemoListBox();
        DemoListBoxAsChild();
    }
}

/*****
*
*           main
*
*****/

void main(void) {
    GUI_Init();
    DemoListbox();
}

```

### Screen shots of above example



## 13.8 PROGBAR: Progress bar widget

Progress bars are commonly used in applications for visualization; for example, a tank fill-level indicator or an oil-pressure indicator. Sample screenshots can be found at the beginning of the chapter and at end of this section. All PROGBAR-related routines are in the file(s) `PROGBAR*.c`, `PROGBAR.h`. All identifiers are prefixed `PROGBAR`.

## Configuration options

Type	Macro	Default	Explanation
S	PROGBAR_DEFAULT_FONT	GUI_DEFAULT_FONT	Font used.
N	PROGBAR_DEFAULT_BARCOLOR0	0x555555 (dark gray)	Left bar color.
N	PROGBAR_DEFAULT_BARCOLOR1	0xAAAAAA (light gray)	Right bar color.
N	PROGBAR_DEFAULT_TEXTCOLOR0	0xFFFFFFFF	Text color, left bar.
N	PROGBAR_DEFAULT_TEXTCOLOR1	0x000000	Text color, right bar.

## PROGBAR API

The table below lists the available  $\mu$ C/GUI PROGBAR-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<a href="#">PROGBAR_Create()</a>	Create the progress bar.
<a href="#">PROGBAR_CreateIndirect()</a>	Create the progress bar from resource table entry.
<a href="#">PROGBAR_SetBarColor()</a>	Sets the color(s) for the bar.
<a href="#">PROGBAR_SetFont()</a>	Select the font for the text.
<a href="#">PROGBAR_SetMinMax()</a>	Set the minimum and maximum values used for the bar.
<a href="#">PROGBAR_SetText()</a>	Set the (optional) text for the bargraph.
<a href="#">PROGBAR_SetTextAlign()</a>	Set text alignment (default is centered).
<a href="#">PROGBAR_SetTextColor()</a>	Set the color(s) for the text.
<a href="#">PROGBAR_SetTextPos()</a>	Set the text position (default 0,0).
<a href="#">PROGBAR_SetValue()</a>	Set the value for the bar graph (and percentage if no text has been assigned).

## PROGBAR\_Create()

### Description

Creates a PROGBAR widget of a specified size at a specified location.

### Prototype

```
PROGBAR_Handle PROGBAR_Create(int x0, int y0, int xsize, int ysize, int
Flags);
```

Parameter	Meaning
<a href="#">x0</a>	Leftmost pixel of the progress bar (in desktop coordinates).
<a href="#">y0</a>	Topmost pixel of the progress bar (in desktop coordinates).
<a href="#">xsize</a>	Horizontal size of the progress bar (in pixels).
<a href="#">ysize</a>	Vertical size of the progress bar (in pixels).
<a href="#">Flags</a>	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 12: "The Window Manager" for a list of available parameter values).

### Return value

Handle for the created PROGBAR widget; 0 if the routine fails.

## PROGBAR\_CreateIndirect()

Prototype explained at the beginning of the chapter. The elements `Flags` and `Para` of the resource passed as parameter are not used.

## PROGBAR\_SetBarColor()

### Description

Sets the color(s) of the progress bar.

### Prototype

```
void PROGBAR_SetBarColor(PROGBAR_Handle hObj, int Index, GUI_COLOR Color);;
```

Parameter	Meaning
<code>hObj</code>	Handle of progress bar.
<code>Index</code>	See table below. Other values are not permitted.
<code>Color</code>	Color to set (24-bit RGB value).

Permitted values for parameter <code>Index</code>	
0	Left portion of the progress bar.
1	Right portion of the progress bar.

## PROGBAR\_SetFont()

### Description

Selects the font for the text display inside the progress bar.

### Prototype

```
void PROGBAR_SetFont(PROGBAR_Handle hObj, const GUI_FONT* pFont);
```

Parameter	Meaning
<code>hObj</code>	Handle of progress bar.
<code>pFont</code>	Pointer to the font.

### Additional information

If this function is not called, the default font for progress bars (the GUI default font) will be used. However, the progress bar default font may be changed in the `GUIConf.h` file.

Simply `#define` the default font as follows (example):

```
#define PROGBAR_DEFAULT_FONT &GUI_Font13_ASCII
```

## PROGBAR\_SetMinMax()

### Description

Sets the minimum and maximum values used for the progress bar.

## Prototype

```
void PROGBAR_SetMinMax(PROGBAR_Handle hObj, int Min, int Max);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of progress bar.
<a href="#">Min</a>	Minimum value Range: $-16383 < \text{Min} \leq 16383$ .
<a href="#">Max</a>	Maximum value Range: $-16383 < \text{Max} \leq 16383$ .

## Additional information

If this function is not called, the default values of [Min](#) = 0, [Max](#) = 100 will be used.

## PROGBAR\_SetText()

### Description

Sets the text displayed inside the progress bar.

### Prototype

```
void PROGBAR_SetText(PROGBAR_Handle hObj, const char* s);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of progress bar.
<a href="#">s</a>	Text to display. A NULL pointer is permitted; in this case a percentage value will be displayed.

## Additional information

If this function is not called, a percentage value will be displayed as the default. If you do not want to display any text at all, you should set an empty string.

## PROGBAR\_SetTextAlign()

### Description

Sets the text alignment.

### Prototype

```
void PROGBAR_SetTextAlign(PROGBAR_Handle hObj, int Align);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of progress bar.
<a href="#">Align</a>	Horizontal alignment attribute for the text (see table below).

Permitted values for parameter <a href="#">Align</a>	
GUI_TA_HCENTER	Centers the title (default).
GUI_TA_LEFT	Displays the title to the left.
GUI_TA_RIGHT	Displays the title to the right.

## Additional information

If this function is not called, the default behavior is to display the text centered.

## PROGBAR\_SetTextPos()

### Description

Sets the text position in pixels.

### Prototype

```
void PROGBAR_SetTextPos(PROGBAR_Handle hObj, int XOff, int YOff);
```

Parameter	Meaning
<code>hObj</code>	Handle of progress bar.
<code>XOff</code>	Number of pixels to move text in horizontal direction. Positive number will move text to the right.
<code>YOff</code>	Number of pixels to move text in vertical direction. Positive number will move text down.

### Additional information

The values move the text the specified number of pixels within the widget. Normally, the default of (0,0) should be sufficient.

## PROGBAR\_SetValue()

### Description

Sets the value of the progress bar.

### Prototype

```
void PROGBAR_SetValue(PROGBAR_Handle hObj, int v);
```

Parameter	Meaning
<code>hObj</code>	Handle of progress bar.
<code>v</code>	Value to set.

### Additional information

The bar indicator will be calculated with regard to the max/min values. If a percentage is automatically displayed, the percentage will also be calculated using the given min/max values as follows:

$$p = 100\% * (v - \text{Min}) / (\text{Max} - \text{Min})$$

The default value after creation of the widget is 0.

## Examples

### Using the PROGBAR widget

The following simple example demonstrates the use of the PROGBAR widget. It is available in the samples as `WIDGET_SimpleProgbar.c`:

```

/*****
*                               Micrium Inc.                               *
*                               Empowering embedded systems               *
*                               uC/GUI sample code                       *
*                               ****                                     ****
*****/

-----
File       : WIDGET_SimpleProgbar.c
Purpose    : Demonstrates the use of the PROGBAR widget

```

```

-----
*/

#include "gui.h"
#include "progbar.h"

/*****
 *
 *           Shows the use of progress bars
 *
 *****/

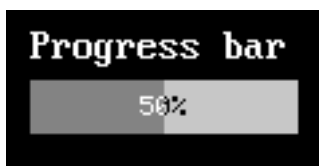
static void DemoProgBar(void) {
    int i;
    PROGBAR_Handle ahProgBar;
    GUI_Clear();
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringAt("Progress bar", 100,80);
    /* Create progress bar */
    ahProgBar = PROGBAR_Create(100, 100, 100, 20, WM_CF_SHOW);
    GUI_Delay (500);
    /* Modify progress bar */
    for (i = 0; i <= 100; i++) {
        PROGBAR_SetValue(ahProgBar, i);
        GUI_Delay(10);
    }
    GUI_Delay (500);
    /* Delete progress bar */
    PROGBAR_Delete(ahProgBar);
}

/*****
 *
 *           main
 *
 *****/

void main(void) {
    GUI_Init();
    while(1) {
        DemoProgBar();
    }
}

```

### Screen shot of above example



### Advanced use of the PROGBAR widget

This more advanced example creates a tank fill-level indicator. It is available as `WIDGET_Progbar.c`:

```

/*****
 *
 *           Micrium Inc.
 *           Empowering embedded systems
 *
 *           µC/GUI sample code
 *
 *****/

-----
File       : WIDGET_Progbar.c
Purpose    : Simple demo shows the use of the PROGBAR widget
-----

```

```

*/

#include "gui.h"
#include "progressbar.h"

/*****
*
*           Shows the use of progress bars
*
*****/

static void DemoProgBar(void) {
    int i;
    PROGBAR_Handle ahProgBar[2];
    GUI_Clear();
    GUI_SetColor(GUI_WHITE);
    GUI_SetFont(&GUI_Font8x16);
    GUI_DispStringAt("Progress bar", 100,80);
    /* Create `em */
    ahProgBar[0] = PROGBAR_Create(100,100,100,20, WM_CF_SHOW);
    ahProgBar[1] = PROGBAR_Create( 80,150,140,10, WM_CF_SHOW);
    /* Use memory device (optional, for better looks) */
    PROGBAR_EnableMemdev(ahProgBar[0]);
    PROGBAR_EnableMemdev(ahProgBar[1]);
    GUI_Delay (1000);
    PROGBAR_SetMinMax(ahProgBar[1], 0, 500);
    while(1) {
        PROGBAR_SetFont(ahProgBar[0], &GUI_Font8x16);
        #if (LCD_BITSPERPIXEL <= 4)
            PROGBAR_SetBarColor(ahProgBar[0], 0, GUI_DARKGRAY);
            PROGBAR_SetBarColor(ahProgBar[0], 1, GUI_LIGHTGRAY);
        #else
            PROGBAR_SetBarColor(ahProgBar[0], 0, GUI_GREEN);
            PROGBAR_SetBarColor(ahProgBar[0], 1, GUI_RED);
        #endif
        for (i=0; i<=100; i++) {
            PROGBAR_SetValue(ahProgBar[0], i);
            PROGBAR_SetValue(ahProgBar[1], i);
            GUI_Delay(5);
        }
        PROGBAR_SetText(ahProgBar[0], "Tank empty");
        for (; i>=0; i--) {
            PROGBAR_SetValue(ahProgBar[0], i);
            PROGBAR_SetValue(ahProgBar[1], 200-i);
            GUI_Delay(5);
        }
        PROGBAR_SetText(ahProgBar[0], "Any text ...");
        PROGBAR_SetTextAlign(ahProgBar[0], GUI_TA_LEFT);
        for (; i<=100; i++) {
            PROGBAR_SetValue(ahProgBar[0], i);
            PROGBAR_SetValue(ahProgBar[1], 200+i);
            GUI_Delay(5);
        }
        PROGBAR_SetTextAlign(ahProgBar[0], GUI_TA_RIGHT);
        for (; i>=0; i--) {
            PROGBAR_SetValue(ahProgBar[0], i);
            PROGBAR_SetValue(ahProgBar[1], 400-i);
            GUI_Delay(5);
        }
        PROGBAR_SetFont(ahProgBar[0], &GUI_FontComic18B_1);
        PROGBAR_SetText(ahProgBar[0], "Any font ...");
        for (; i<=100; i++) {
            PROGBAR_SetValue(ahProgBar[0], i);
            PROGBAR_SetValue(ahProgBar[1], 400+i);
            GUI_Delay(5);
        }
        GUI_Delay(1000);
    }
}

/*****
*

```

```
*
*                               main
*
*****
*/

void main(void) {
    GUI_Init();
    DemoProgBar();
}
```

Screen shot of above example



13.9 RADIO: Radio button widget

Radio buttons, like check boxes, are used for selecting choices. A dot appears when a radio button is turned on or selected. The difference from check boxes is that the user can only select one radio button at a time. When a button is selected, the other buttons in the widget are turned off, as shown to the right. One radio button widget may contain any number of buttons, which are always arranged vertically. All RADIO-related routines are located in the file(s) `RADIO*.c`, `RADIO.h`. All identifiers are prefixed `RADIO`.



Configuration options

Type	Macro	Default	Explanation
	<code>RADIO_BKCOLOR0_DEFAULT</code>	<code>0xc0c0c0</code>	Background color of button, inactive state.
	<code>RADIO_BKCOLOR1_DEFAULT</code>	<code>GUI_WHITE</code>	Background color of button, active state.

RADIO API

The table below lists the available  $\mu$ C/GUI RADIO-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<a href="#">RADIO_Create()</a>	Create a group of radio buttons.
<a href="#">RADIO_CreateIndirect()</a>	Create a group of radio buttons from resource table entry.
<a href="#">RADIO_Dec()</a>	Decrement the button selection by a value of 1.
<a href="#">RADIO_GetValue()</a>	Return the current button selection.
<a href="#">RADIO_Inc()</a>	Increment the button selection by a value of 1.
<a href="#">RADIO_SetValue()</a>	Set the button selection.

## RADIO\_Create()

### Description

Creates a RADIO widget of a specified size at a specified location.

### Prototype

```
RADIO_Handle RADIO_Create(int x0, int y0, int xsize, int ysize, WM_HWIN
                          hParent, int Id, int Flags, unsigned Para);
```

Parameter	Meaning
<code>x0</code>	Leftmost pixel of the radio button widget (in desktop coordinates).
<code>y0</code>	Topmost pixel of the radio button widget (in desktop coordinates).
<code>xsize</code>	Horizontal size of the radio button widget (in pixels).
<code>ysize</code>	Vertical size of the radio button widget (in pixels).
<code>hParent</code>	Handle of parent window.
<code>Id</code>	ID to be returned.
<code>Flags</code>	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 12: "The Window Manager" for a list of available parameter values).
<code>Para</code>	Number of buttons in the group.

### Return value

Handle for the created RADIO widget; 0 if the routine fails.

## RADIO\_CreateIndirect()

Prototype explained at the beginning of the chapter. The element `Flags` of the resource passed as parameter is not used, but the number of radio buttons used is specified as the `Para` element.

## RADIO\_Dec()

### Description

Decrements the selection by a value of 1.

### Prototype

```
void RADIO_Dec(RADIO_Handle hObj);
```

Parameter	Meaning
<code>hObj</code>	Handle of radio button widget.

### Additional information

Please note that the numbering of the buttons always starts from the top with a value of 0; therefore decrementing the selection will actually move the selection one button up.

## RADIO\_GetValue()

### Description

Returns the current button selection.

## Prototype

```
void RADIO_GetValue(RADIO_Handle hObj);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of radio button widget.

## Return value

The value of the currently selected button.

## RADIO\_Inc()

### Description

Increments the selection by a value of 1.

### Prototype

```
void RADIO_Inc(RADIO_Handle hObj);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of radio button widget.

### Additional information

Please note that the numbering of the buttons always starts from the top with a value of 0; therefore incrementing the selection will actually move the selection one button down.

## RADIO\_SetValue()

### Description

Sets the current button selection.

### Prototype

```
void RADIO_SetValue(RADIO_Handle hObj, int v);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of radio button widget.
<a href="#">v</a>	Value to be set.

### Additional information

The topmost radio button in a RADIO widget always has the 0 value, the next button down is always 1, the next is 2, etc.

## 13.10 SCROLLBAR: Scroll bar widget

Scroll bars are used for scrolling through list boxes or any windows which do not fit entirely into their frames. They may be created horizontally, as shown below, or vertically.



A scroll bar is typically attached to an existing window, for example the list box shown below:



All SCROLLBAR-related routines are located in the file(s) `SCROLLBAR*.c`, `SCROLLBAR.h`. All identifiers are prefixed `SCROLLBAR`.

## Configuration options

Type	Macro	Default	Explanation
N	<code>SCROLLBAR_BKCOLOR0_DEFAULT</code>	0x808080	Background (thumb area) color.
N	<code>SCROLLBAR_BKCOLOR1_DEFAULT</code>	<code>GUI_BLACK</code>	Scroll bar (thumb) color.
N	<code>SCROLLBAR_COLOR0_DEFAULT</code>	0xc0c0c0	Arrow button colors.
B	<code>SCROLLBAR_USE_3D</code>	1	Enable 3D support.

## SCROLLBAR API

The table below lists the available  $\mu$ C/GUI SCROLLBAR-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<a href="#"><code>SCROLLBAR_AddValue()</code></a>	Increment or decrement the value of the scroll bar by a specified value.
<a href="#"><code>SCROLLBAR_Create()</code></a>	Create the scroll bar.
<a href="#"><code>SCROLLBAR_CreateAttached()</code></a>	Create a scroll bar attached to a window.
<a href="#"><code>SCROLLBAR_CreateIndirect()</code></a>	Create the scroll bar from resource table entry.
<a href="#"><code>SCROLLBAR_Dec()</code></a>	Decrement the value of the scroll bar by a value of 1.
<a href="#"><code>SCROLLBAR_GetValue()</code></a>	Return the current item value.
<a href="#"><code>SCROLLBAR_Inc()</code></a>	Increment the value of the scroll bar by a value of 1.
<a href="#"><code>SCROLLBAR_SetNumItems()</code></a>	Set the number of items for scrolling.
<a href="#"><code>SCROLLBAR_SetPageSize()</code></a>	Set the page size (in number of items).
<a href="#"><code>SCROLLBAR_SetValue()</code></a>	Set the current value of the scroll bar.
<a href="#"><code>SCROLLBAR_SetWidth()</code></a>	Set the width of the scroll bar.

## SCROLLBAR\_AddValue()

### Definition

Increments or decrements the value of the scroll bar by a specified value.

### Prototype

```
void SCROLLBAR_AddValue(SCROLLBAR_Handle hObj, int Add);
```

Parameter	Meaning
<a href="#"><code>hObj</code></a>	Handle of scroll bar.
<a href="#"><code>Add</code></a>	Number of items to increment or decrement at one time.

**Additional information**

The scroll bar cannot exceed the value set in `SCROLLBAR_SetNumItems()`. For example, if a window contains 200 items and the scroll bar is currently at value 195, incrementing the bar by 3 items will move it to value 198. However, incrementing by 10 items will only move the bar as far as value 200, which is the maximum value for this particular window.

**SCROLLBAR\_Create()**

**Description**

Creates a SCROLLBAR widget of a specified size at a specified location.

**Prototype**

```
SCROLLBAR_Handle SCROLLBAR_Create(int x0, int y0, int xsize, int ysize
                                WM_HWIN hParent, int Id, int WinFlags,
                                int SpecialFlags);
```

Parameter	Meaning
x0	Leftmost pixel of the scroll bar (in desktop coordinates).
y0	Topmost pixel of the scroll bar (in desktop coordinates).
xsize	Horizontal size of the scroll bar (in pixels).
ysize	Vertical size of the scroll bar (in pixels).
hParent	Handle of parent window.
Id	ID to be returned.
WinFlags	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow( ) in Chapter 12: "The Window Manager" for a list of available parameter values).
SpecialFlags	Special creation flags (see indirect creation flags under SCROLLBAR_CreateIndirect()).

**Return value**

Handle for the created SCROLLBAR widget; 0 if the routine fails.

**SCROLLBAR\_CreateAttached()**

**Description**

Creates a scroll bar which is attached to an existing window.

**Prototype**

```
SCROLLBAR_Handle SCROLLBAR_CreateAttached(WM_HWIN hParent,
                                           int SpecialFlags);
```

Parameter	Meaning
hParent	Handle of parent window.
SpecialFlags	Special creation flags (see indirect creation flags under SCROLLBAR_CreateIndirect()).

**Return value**

Handle for the created scrollbar; 0 if the routine fails.

## Additional information

An attached scroll bar is essentially a child window which will position itself on the parent window and operate accordingly.

Vertical attached scrollbars will be automatically placed on the right side of the parent window; horizontal scrollbars on the bottom.

## Example

Creates a list box with an attached scrollbar:

```
LISTBOX_Handle hListBox;
hListBox = LISTBOX_Create(ListBox, 50, 50, 100, 100, WM_CF_SHOW);
SCROLLBAR_CreateAttached(hListBox, SCROLLBAR_CF_VERTICAL);
```

## Screen shots of above example

The picture on the left shows the list box as it appears after creation. On the right it is shown with the attached vertical scrollbar:



## SCROLLBAR\_CreateIndirect()

Prototype explained at the beginning of the chapter. The following flags may be used as the `Flags` element of the resource passed as parameter:

Permitted indirect creation flags ("OR" combinable)	
SCROLLBAR_CF_VERTICAL	Create a vertical scroll bar (default is horizontal).
SCROLLBAR_CF_FOCUSSABLE	Gives scroll bar the input focus.

The `Para` element is not used in the resource table.

## SCROLLBAR\_Dec()

### Description

Decrements the current value of the scroll bar by a value of 1.

### Prototype

```
void SCROLLBAR_Dec(SCROLLBAR_Handle hObj);
```

Parameter	Meaning
<code>hObj</code>	Handle of scroll bar.

## Additional information

The definition of an "item" is application-specific, although in most cases it is equal to one line. Items are numbered top to bottom or left to right, beginning with a value of 0.

## SCROLLBAR\_GetValue()

### Description

Return the value of the current item.

## Prototype

```
int SCROLLBAR_GetValue(SCROLLBAR_Handle hObj);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of scroll bar.

## Return value

The value of the current item.

## SCROLLBAR\_Inc()

### Description

Increments the current value of the scroll bar by a value of 1.

### Prototype

```
void SCROLLBAR_Inc(SCROLLBAR_Handle hObj);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of scroll bar.

### Additional information

The definition of an "item" is application-specific, although in most cases it is equal to one line. Items are numbered top to bottom or left to right, beginning with a value of 0.

## SCROLLBAR\_SetNumItems()

### Description

Sets the number of items for scrolling.

### Prototype

```
void SCROLLBAR_SetNumItems(SCROLLBAR_Handle hObj, int NumItems);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of scroll bar.
<a href="#">NumItems</a>	Number of items to be set.

### Additional information

The definition of an "item" is application-specific, although in most cases it is equal to one line.

The number of items specified is the maximum value; the scroll bar cannot go beyond this value.

## SCROLLBAR\_SetPageSize()

### Description

Sets the page size.

## Prototype

```
void SCROLLBAR_SetPageSize(SCROLLBAR_Handle hObj, int PageSize);
```

Parameter	Meaning
<code>hObj</code>	Handle of scroll bar.
<code>PageSize</code>	Page size (in number of items).

## Additional information

Page size is specified as the number of items to one page. If the user pages up or down, either with the keyboard or by mouse-clicking in the scroll bar area, the window will be scrolled up or down by the number of items specified to be one page.

## SCROLLBAR\_SetValue()

### Description

Sets the current value of the scroll bar.

### Prototype

```
void SCROLLBAR_SetValue(SCROLLBAR_Handle hObj, int v);
```

Parameter	Meaning
<code>hObj</code>	Handle of scroll bar.
<code>v</code>	Value to be set.

## SCROLLBAR\_SetWidth()

### Description

Sets the width of the scroll bar.

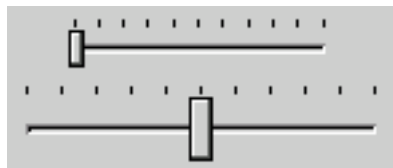
### Prototype

```
void SCROLLBAR_SetWidth(SCROLLBAR_Handle hObj, int Width);
```

Parameter	Meaning
<code>hObj</code>	Handle of scroll bar.
<code>Width</code>	Width to be set.

## 13.11 SLIDER: Slider widget

Slider widgets are commonly used for modifying values through the use of a slider bar.



All SLIDER-related routines are located in the file(s) `SLIDER*.c`, `SLIDER.h`. All identifiers are prefixed `SLIDER`.

## Configuration options

Type	Macro	Default	Explanation
N	SLIDER_BKCOLOR0_DEFAULT	0xc0c0c0	Background color.
N	SLIDER_COLOR0_DEFAULT	0xc0c0c0	Slider (thumb) color.
B	SLIDER_USE_3D	1	Enable 3D support.

## SLIDER API

The table below lists the available  $\mu$ C/GUI SLIDER-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<a href="#">SLIDER_Create()</a>	Create the slider.
<a href="#">SLIDER_CreateIndirect()</a>	Create the slider from resource table entry.
<a href="#">SLIDER_Dec()</a>	Decrement the value of the slider bar.
<a href="#">SLIDER_GetValue()</a>	Return the current value of the slider bar.
<a href="#">SLIDER_Inc()</a>	Increment the value of the slider bar.
<a href="#">SLIDER_SetRange()</a>	Set the range of the slider value.
<a href="#">SLIDER_SetValue()</a>	Set the current value of the slider bar.
<a href="#">SLIDER_SetWidth()</a>	Set the width of the slider bar.

## SLIDER\_Create()

### Description

Creates a SLIDER widget of a specified size at a specified location.

### Prototype

```
SLIDER_Handle SLIDER_Create(int x0, int y0,
                             int xsize, int ysize,
                             WM_HWIN hParent, int Id, int WinFlags,
                             int SpecialFlags);
```

Parameter	Meaning
<a href="#">x0</a>	Leftmost pixel of the slider (in desktop coordinates).
<a href="#">y0</a>	Topmost pixel of the slider (in desktop coordinates).
<a href="#">xsize</a>	Horizontal size of the slider (in pixels).
<a href="#">ysize</a>	Vertical size of the slider (in pixels).
<a href="#">hParent</a>	Handle of the parent window.
<a href="#">Id</a>	Id to be returned
<a href="#">WinFlags</a>	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 12: "The Window Manager" for a list of available parameter values).
<a href="#">SpecialFlags</a>	Special creation flag (see indirect creation flag under SLIDER_CreateIndirect()).

### Return value

Handle for the created SLIDER widget; 0 if the routine fails.

## SLIDER\_CreateIndirect()

Prototype explained at the beginning of the chapter. The following flag may be used as the `Flags` element of the resource passed as parameter:

Permitted indirect creation flag	
SLIDER_CF_VERTICAL	Create a vertical slider (default is horizontal).

The `Para` element is not used in the resource table.

## SLIDER\_Dec()

### Description

Decrements the current value of the slider bar by one item.

### Prototype

```
void SLIDER_Dec(SLIDER_Handle hObj);
```

Parameter	Meaning
<code>hObj</code>	Handle of slider widget.

## SLIDER\_GetValue()

### Description

Returns the current value of the slider bar.

### Prototype

```
int SLIDER_GetValue(SLIDER_Handle hObj);
```

Parameter	Meaning
<code>hObj</code>	Handle of slider widget.

### Return value

The current value of the slider.

## SLIDER\_Inc()

### Description

Increments the current value of the slider bar by one item.

### Prototype

```
void SLIDER_Inc(SLIDER_Handle hObj);
```

Parameter	Meaning
<code>hObj</code>	Handle of slider widget.

## SLIDER\_SetRange()

### Description

Sets the range of the slider.

## Prototype

```
void SLIDER_SetRange(SLIDER_Handle hObj, int Min, int Max);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of slider widget.
<a href="#">Min</a>	Minimum value.
<a href="#">Max</a>	Maximum value.

## Additionnal information

The default values after creating a slider are 0 - 100.

## SLIDER\_SetValue()

### Description

Sets the current value of the slider bar.

### Prototype

```
void SLIDER_SetValue(SLIDER_Handle hObj, int v);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of slider widget.
<a href="#">v</a>	Value to be set.

## SLIDER\_SetWidth()

### Description

Sets the width of the slider bar.

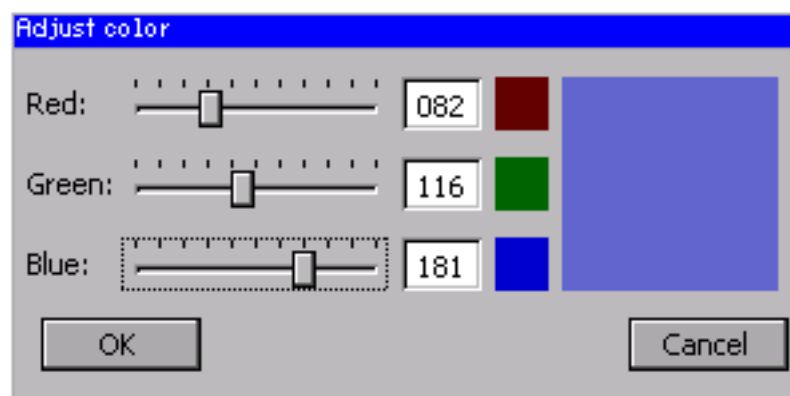
### Prototype

```
void SLIDER_SetWidth(SLIDER_Handle hObj, int Width);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of slider widget.
<a href="#">Width</a>	Width to be set.

## Example

The source of the following sample is available as `DIALOG_SliderColor.c` in the samples shipped with `μC/GUI`:

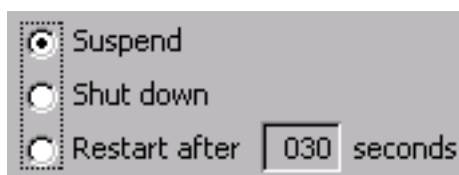


## 13.12 TEXT: Text widget

Text widgets are typically used in order to display fields of text in dialog boxes, as shown in the message box below:



Of course, text fields may also be used for labeling other widgets, as follows:



All TEXT-related routines are located in the file(s) `TEXT*.c`, `TEXT.h`. All identifiers are prefixed TEXT.

## Configuration options

Type	Macro	Default	Explanation
S	TEXT_FONT_DEFAULT	&GUI_Font13_1	Font used.

## TEXT API

The table below lists the available  $\mu$ C/GUI TEXT-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<a href="#">TEXT_Create()</a>	Create the text.
<a href="#">TEXT_CreateIndirect()</a>	Create the text from resource table entry.
<a href="#">TEXT_GetDefaultFont()</a>	Return the default font used for text.
<a href="#">TEXT_SetDefaultFont()</a>	Set the default font used for text.
<a href="#">TEXT_SetFont()</a>	Set the font used for a specified text widget.
<a href="#">TEXT_SetText()</a>	Set the text for a specified text widget.
<a href="#">TEXT_SetTextAlign()</a>	Set the text alignment of a specified text widget.
<a href="#">TEXT_SetTextPos()</a>	Set the position of the text in a specified text widget.

## TEXT\_Create()

### Description

Creates a TEXT widget of a specified size at a specified location.

### Prototype

```
TEXT_Handle TEXT_Create(int x0, int y0,
                        int xsize, int ysize,
                        int Id, int Flags,
                        const char* s, int Align);
```

Parameter	Meaning
<a href="#">x0</a>	Leftmost pixel of the slider (in desktop coordinates).
<a href="#">y0</a>	Topmost pixel of the slider (in desktop coordinates).
<a href="#">xsize</a>	Horizontal size of the slider (in pixels).
<a href="#">ysize</a>	Vertical size of the slider (in pixels).
<a href="#">Id</a>	ID to be returned.
<a href="#">Flags</a>	Window create flags. Typically WM_CF_SHOW in order to make the widget visible immediately (please refer to WM_CreateWindow() in Chapter 12: "The Window Manager" for a list of available parameter values).
<a href="#">s</a>	Pointer to the text to be displayed.
<a href="#">Align</a>	Alignment attribute for the text (see indirect creation flags under TEXT_CreateIndirect()).

### Return value

Handle for the created TEXT widget; 0 if the routine fails.

## TEXT\_CreateIndirect()

Prototype explained at the beginning of the chapter. The following flags may be used as the `Flags` element of the resource passed as parameter:

Permitted indirect creation flags ("OR" combinable)	
TEXT_CF_LEFT	Horizontal alignment: left
TEXT_CF_RIGHT	Horizontal alignment: right
TEXT_CF_HCENTER	Horizontal alignment: center
TEXT_CF_TOP	Vertical alignment: top
TEXT_CF_BOTTOM	Vertical alignment: bottom
TEXT_CF_VCENTER	Vertical alignment: center

The `Para` element is not used in the resource table.

## TEXT\_GetDefaultFont()

### Description

Retruns the default font used for text widgets.

### Prototype

```
const GUI_FONT* TEXT_GetDefaultFont(void);
```

### Return value

Pointer to the default font used for text widgets.

## TEXT\_SetDefaultFont()

### Description

Sets the default font used for text widgets.

### Prototype

```
void TEXT_SetDefaultFont(const GUI_FONT* pFont);
```

Parameter	Meaning
<code>pFont</code>	Pointer to the font to be set as default.

## TEXT\_SetFont()

### Description

Sets the font to be used for a specified text widget.

### Prototype

```
void TEXT_SetFont(TEXT_Handle hObj, const GUI_FONT* pFont);
```

Parameter	Meaning
<code>hObj</code>	Handle of text widget.
<code>pFont</code>	Pointer to the font to be used.

## TEXT\_SetText()

### Description

Sets the text to be used for a specified text widget.

### Prototype

```
void TEXT_SetText(TEXT_Handle hObj, const char* s);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of text widget.
<a href="#">s</a>	Text to be displayed.

## TEXT\_SetTextAlign()

### Description

Sets the text alignment of a specified text widget.

### Prototype

```
void TEXT_SetTextAlign(TEXT_Handle hObj, int Align);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of text widget.
<a href="#">Align</a>	Text alignment (see <code>TEXT_Create()</code> ).

## TEXT\_SetTextPos()

### Description

Sets the position of the text in a specified text widget.

### Prototype

```
void TEXT_SetTextPos(TEXT_Handle hObj, int XOff, int YOff);
```

Parameter	Meaning
<a href="#">hObj</a>	Handle of text widget.
<a href="#">XOff</a>	X-distance to offset text from origin of widget.
<a href="#">YOff</a>	Y-distance to offset text from origin of widget.

# Chapter 14

## Dialogs

---

Widgets may be created and used on their own, as they are by nature windows themselves. However, it is often desirable to use dialog boxes, which are windows that contain one or more widgets.

A dialog box (or dialog) is normally a window that appears in order to request input from the user. It may contain multiple widgets, requesting information from the user through various selections, or it may take the form of a message box which simply provides information (such as a note or warning to the user) and an "OK" button.

## 14.1 Dialog basics

### Input focus

The window manager remembers the window or window object that was last selected by the user with the touch-screen, mouse, keyboard, or other means. This window receives keyboard input messages and is said to have the input focus.

The primary reason for keeping track of input focus is to determine where to send keyboard commands. The window which has input focus will receive events generated by the keyboard.

### Blocking vs. non-blocking dialogs

Dialog windows can be blocking or non-blocking. A blocking dialog blocks the thread of execution. It has input focus by default and must be closed by the user before the thread can continue. A non-blocking dialog, on the other hand, does not block the calling thread -- it allows the task to continue while it is displayed.

Please note that in some other environments, blocking and non-blocking dialogs may be referred to as modal and modeless, respectively.

### Dialog messages

A dialog box is a window, and it receives messages just like all other windows in the system do. Most messages are handled by the dialog box automatically; the others are passed to the callback routine specified upon creation of the dialog box (also known as the dialog procedure).

There are two types of additional messages which are sent to the dialog window procedure: `WM_INIT_DIALOG` and `WM_NOTIFY_PARENT`. The `WM_INIT_DIALOG` message is sent to the dialog box procedure immediately before a dialog box is displayed. Dialog procedures typically use this message to initialize widgets and carry out any other initialization tasks that affect the appearance of the dialog box. The `WM_NOTIFY_PARENT` message is sent to the dialog box by its child windows in order to notify the parent of any events in order to ensure synchronization. The events sent by a child depend on its type and are documented separately for every type of widget.

## 14.2 Creating a dialog

Two basic things are required to create a dialog box: a resource table that defines the widgets to be included, and a dialog procedure which defines the initial values for the widgets as well as their behavior. Once both items exist, you need only a single function call (`GUI_CreateDialogBox()` or `GUI_ExecDialogBox()`) to actually create the dialog.

### Resource table

Dialog boxes may be created in a blocking manner (using `GUI_ExecDialogBox()`) or as non-blocking (using `GUI_CreateDialogBox()`). A resource table must first be defined which specifies all widgets to be included in the dialog. The example shown below creates a resource table. This particular sample was created manually, although it could also be done by a GUI-builder:

```
static const GUI_WIDGET_CREATE_INFO _aDialogCreate[] = {
    { FRAMEWIN_CreateIndirect, "Dialog", 0, 10, 10, 180, 230, 0, 0 },
    { BUTTON_CreateIndirect, "OK", GUI_ID_OK, 100, 5, 60, 20, 0, 0 },
    { BUTTON_CreateIndirect, "Cancel", GUI_ID_CANCEL, 100, 30, 60, 20, 0, 0 },
    { TEXT_CreateIndirect, "LText", 0, 10, 55, 48, 15, 0, GUI_TA_LEFT },
    { TEXT_CreateIndirect, "RText", 0, 10, 80, 48, 15, 0, GUI_TA_RIGHT },
    { EDIT_CreateIndirect, NULL, GUI_ID_EDIT0, 60, 55, 100, 15, 0, 50 },
    { EDIT_CreateIndirect, NULL, GUI_ID_EDIT1, 60, 80, 100, 15, 0, 50 },
    { TEXT_CreateIndirect, "Hex", 0, 10, 100, 48, 15, 0, GUI_TA_RIGHT },
    { EDIT_CreateIndirect, NULL, GUI_ID_EDIT2, 60, 100, 100, 15, 0, 6 },
    { TEXT_CreateIndirect, "Bin", 0, 10, 120, 48, 15, 0, GUI_TA_RIGHT },
    { EDIT_CreateIndirect, NULL, GUI_ID_EDIT3, 60, 120, 100, 15, 0, 0 },
    { LISTBOX_CreateIndirect, NULL, GUI_ID_LISTBOX0, 10, 20, 48, 40, 0, 0 },
    { CHECKBOX_CreateIndirect, NULL, GUI_ID_CHECK0, 10, 5, 0, 0, 0, 0 },
    { CHECKBOX_CreateIndirect, NULL, GUI_ID_CHECK1, 30, 5, 0, 0, 0, 0 },
    { SLIDER_CreateIndirect, NULL, GUI_ID_SLIDER0, 60, 140, 100, 20, 0, 0 },
    { SLIDER_CreateIndirect, NULL, GUI_ID_SLIDER1, 10, 170, 150, 30, 0, 0 }
};
```

## Dialog procedure

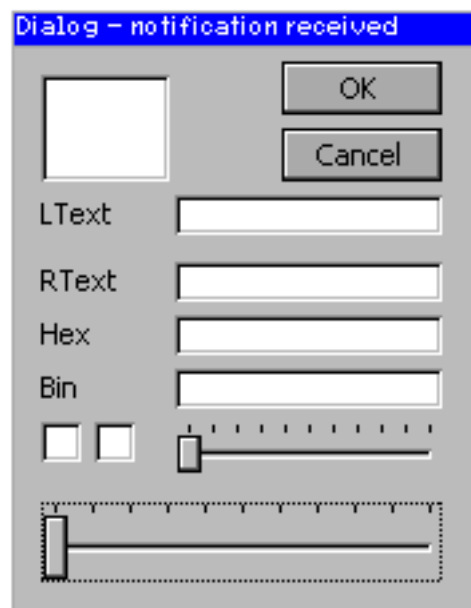
The sample above has been created using the blank dialog procedure shown below. This is the basic template which should be used as a starting point when creating any dialog procedure:

```
/******
 *
 *      Dialog procedure
 */
static void _cbCallback(WM_MESSAGE * pMsg) {
    switch (pMsg->MsgId) {
        default:
            WM_DefaultProc(pMsg);
    }
}
```

For this sample, the dialog box is displayed with the following line of code:

```
GUI_ExecDialogBox(_aDialogCreate, GUI_COUNTOF(_aDialogCreate), &_cbCallback, 0, 0, 0);
```

The resulting dialog box looks as follows, or similar (the actual appearance will depend on your configuration and default settings):



After creation of the dialog box, all widgets included in the resource table will be visible, although as can be seen in the previous screen shot, they will appear "empty". This is because the dialog procedure does not yet contain code that initializes the individual elements. The initial values of the widgets, the actions caused by them, and the interactions between them need to be defined in the dialog procedure.

### Initializing the dialog

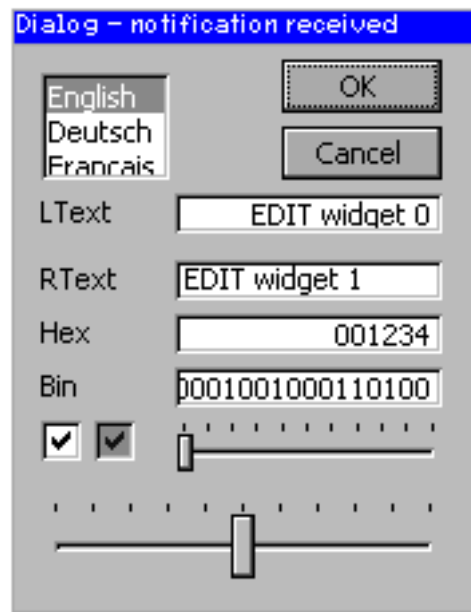
The typical next step is to initialize the widgets with their respective initial values. This is normally done in the dialog procedure as a reaction to the WM\_INIT\_DIALOG message. The program excerpt below illustrates things:

```

/*****
*
*       Dialog procedure
*/
static void _cbCallback(WM_MESSAGE * pMsg) {
    int NCode, Id;
    WM_HWIN hEdit0, hEdit1, hEdit2, hEdit3, hListBox;
    WM_HWIN hWin = pMsg->hWin;
    switch (pMsg->MsgId) {
        case WM_INIT_DIALOG:
            /* Get window handles for all widgets */
            hEdit0 = WM_GetDialogItem(hWin, GUI_ID_EDIT0);
            hEdit1 = WM_GetDialogItem(hWin, GUI_ID_EDIT1);
            hEdit2 = WM_GetDialogItem(hWin, GUI_ID_EDIT2);
            hEdit3 = WM_GetDialogItem(hWin, GUI_ID_EDIT3);
            hListBox = WM_GetDialogItem(hWin, GUI_ID_LISTBOX0);
            /* Initialize all widgets */
            EDIT_SetText(hEdit0, "EDIT widget 0");
            EDIT_SetText(hEdit1, "EDIT widget 1");
            EDIT_SetTextAlign(hEdit1, GUI_TA_LEFT);
            EDIT_SetHexMode(hEdit2, 0x1234, 0, 0xffff);
            EDIT_SetBinMode(hEdit3, 0x1234, 0, 0xffff);
            LISTBOX_SetText(hListBox, _apListBox);
            WM_DisableWindow (WM_GetDialogItem(hWin, GUI_ID_CHECK1));
            CHECKBOX_Check( WM_GetDialogItem(hWin, GUI_ID_CHECK0));
            CHECKBOX_Check( WM_GetDialogItem(hWin, GUI_ID_CHECK1));
            SLIDER_SetWidth( WM_GetDialogItem(hWin, GUI_ID_SLIDER0), 5);
            SLIDER_SetValue( WM_GetDialogItem(hWin, GUI_ID_SLIDER1), 50);
            break;
        default:
            WM_DefaultProc(pMsg);
    }
}

```

The initialized dialog box now appears as follows, with all widgets containing their initial values:



## Defining dialog behavior

Once the dialog has been initialized, all that remains is to add code to the dialog procedure which will define the behavior of the widgets, making them fully operable. Continuing with the same example, the final dialog procedure is shown below:

```

/*****
*
*       Dialog procedure
*/
static void _cbCallback(WM_MESSAGE * pMsg) {
    int NCode, Id;
    WM_HWIN hEdit0, hEdit1, hEdit2, hEdit3, hListBox;
    WM_HWIN hWin = pMsg->hWin;
    switch (pMsg->MsgId) {
        case WM_INIT_DIALOG:
            /* Get window handles for all widgets */
            hEdit0 = WM_GetDialogItem(hWin, GUI_ID_EDIT0);
            hEdit1 = WM_GetDialogItem(hWin, GUI_ID_EDIT1);
            hEdit2 = WM_GetDialogItem(hWin, GUI_ID_EDIT2);
            hEdit3 = WM_GetDialogItem(hWin, GUI_ID_EDIT3);
            hListBox = WM_GetDialogItem(hWin, GUI_ID_LISTBOX0);
            /* Initialize all widgets */
            EDIT_SetText(hEdit0, "EDIT widget 0");
            EDIT_SetText(hEdit1, "EDIT widget 1");
            EDIT_SetTextAlign(hEdit1, GUI_TA_LEFT);
            EDIT_SetHexMode(hEdit2, 0x1234, 0, 0xffff);
            EDIT_SetBinMode(hEdit3, 0x1234, 0, 0xffff);
            LISTBOX_SetText(hListBox, _apListBox);
            WM_DisableWindow(WM_GetDialogItem(hWin, GUI_ID_CHECK1));
            CHECKBOX_Check(WM_GetDialogItem(hWin, GUI_ID_CHECK0));
            CHECKBOX_Check(WM_GetDialogItem(hWin, GUI_ID_CHECK1));
            SLIDER_SetWidth(WM_GetDialogItem(hWin, GUI_ID_SLIDER0), 5);
            SLIDER_SetValue(WM_GetDialogItem(hWin, GUI_ID_SLIDER1), 50);
            break;
        case WM_KEY:
            switch (((WM_KEY_INFO*)(pMsg->Data.p))>Key) {
                case GUI_ID_ESCAPE:
                    GUI_EndDialog(hWin, 1);
                    break;
                case GUI_ID_ENTER:
                    GUI_EndDialog(hWin, 0);
            }
    }
}

```

```
        break;
    }
    break;
case WM_NOTIFY_PARENT:
    Id      = WM_GetId(pMsg->hWinSrc);    /* Id of widget */
    NCode = pMsg->Data.v;                 /* Notification code */
    switch (NCode) {
        case WM_NOTIFICATION_RELEASED:    /* React only if released */
            if (Id == GUI_ID_OK) {        /* OK Button */
                GUI_EndDialog(hWin, 0);
            }
            if (Id == GUI_ID_CANCEL) {     /* Cancel Button */
                GUI_EndDialog(hWin, 1);
            }
            break;
        case WM_NOTIFICATION_SEL_CHANGED: /* Selection changed */
            FRAMEWIN_SetText(hWin, "Dialog - sel changed");
            break;
        default:
            FRAMEWIN_SetText(hWin, "Dialog - notification received");
    }
    break;
default:
    WM_DefaultProc(pMsg);
}
}
```

For further details, this entire example is available as `Dialog.c` in the samples shipped with  $\mu$ C/GUI.

### 14.3 API reference: dialogs

The table below lists the available dialog-related routines in alphabetical order within their respective categories. Detailed descriptions of the routines can be found in the sections that follow:

Routine	Explanation
Dialog boxes	
<a href="#">GUI_CreateDialogBox</a>	Create a non-blocking dialog.
<a href="#">GUI_ExecDialogBox</a>	Create a blocking dialog.
<a href="#">GUI_EndDialog</a>	End a dialog box.
Message boxes	
<a href="#">GUI_MessageBox</a>	Create a message box.

### 14.4 Dialog boxes

#### GUI\_CreateDialogBox

**Description**

Creates a non-blocking dialog box.

**Prototype**

```
WM_HWIN GUI_CreateDialogBox(const GUI_WIDGET_CREATE_INFO* paWidget,
                             int NumWidgets, WM_CALLBACK* cb,
```

```
WM_HWIN hParent, int x0, int y0);
```

Parameter	Meaning
<a href="#">paWidget</a>	Pointer to resource table defining the widgets to be included in the dialog.
<a href="#">NumWidgets</a>	Total number of widgets included in the dialog.
<a href="#">cb</a>	Pointer to an application-specific callback function (dialog procedure).
<a href="#">hParent</a>	Handle of parent window (0 = no parent window).
<a href="#">x0</a>	X-position of the dialog relative to parent window.
<a href="#">y0</a>	Y-position of the dialog relative to parent window.

## GUI\_ExecDialogBox

### Description

Creates a blocking dialog box.

### Prototype

```
int GUI_ExecDialogBox(const GUI_WIDGET_CREATE_INFO* paWidget,
                     int NumWidgets, WM_CALLBACK* cb,
                     WM_HWIN hParent, int x0, int y0);
```

Parameter	Meaning
<a href="#">paWidget</a>	Pointer to a resource table defining the widgets to be included in the dialog.
<a href="#">NumWidgets</a>	Total number of widgets included in the dialog.
<a href="#">cb</a>	Pointer to an application-specific callback function (dialog procedure).
<a href="#">hParent</a>	Handle of parent window (0 = no parent window).
<a href="#">x0</a>	X-position of the dialog relative to parent window.
<a href="#">y0</a>	Y-position of the dialog relative to parent window.

## GUI\_EndDialog

### Description

Ends (closes) a dialog box.

### Prototype

```
void GUI_EndDialog(WM_HWIN hDialog, int r);
```

Parameter	Meaning
<a href="#">hDialog</a>	Handle to dialog box.
<a href="#">r</a>	Value to be returned by GUI_ExecDialogBox.

### Return value

Specifies the value to be returned to the calling thread from the function that created the dialog box (typically only relevant with `GUI_ExecDialogBox`). With non-blocking dialogs, there is no application thread waiting and the return value is ignored.

## 14.5 Message boxes

A message box is actually a type of dialog which, once its default properties are specifed, requires only one line of code to create. A message is displayed in a frame window with a title bar, as well as an "OK" button which must be pressed in order to close the window.

### GUI\_MessageBox

#### Description

Creates and displays a message box.

#### Prototype

```
void GUI_MessageBox(const char* sMessage, const char* sCaption, int Flags);
```

Parameter	Meaning
sMessage	Message to display.
sCaption	Caption for the title bar of the frame window.
Flags	Reserved for future extensions, value does not matter.

#### Additionaln information

The default properties of a message box can be changed by modifying them in the GUIConf.h file. The following table lists all available configuration macros:

Type	Macro	Default	Explanation
N	MESSAGEBOX_BORDER	4	Distance between the elements of a message box and the elements of the client window frame.
N	MESSAGEBOX_XSIZEOK	50	X-size of the "OK" button.
N	MESSAGEBOX_YSIZEOK	20	Y-size of the "OK" button.
S	MESSAGEBOX_BKCOLOR	GUI_WHITE	Color of the client window background.

It is possible to display messages with more than one row. The example below shows how to do this.

### Example

The following example demonstrates the use of a simple message box:

```

/*****
*                               Micrium Inc.                               *
*           Empowering embedded systems                                   *
*                               uC/GUI sample code                       *
*                               ****                                     ****
*****/

-----
File      : DIALOG_MessageBox.c
Purpose   : Example demonstrating GUI_MessageBox
-----
*/

#include "GUI.h"

/*****
*
*****/
```

```

*                               main
*
*****
*/

void main(void) {
    GUI_Init();
    WM_SetBkWindowColor(GUI_RED);
    /* Create message box and wait until it is closed */
    GUI_MessageBox("This text is shown\n\n a message box",
                  "Caption/Title", GUI_MESSAGEBOX_CF_MOVEABLE);
    GUI_Delay(500);                               /* Wait for a short moment ... */
    GUI_MessageBox("New message !",
                  "Caption/Title", GUI_MESSAGEBOX_CF_MOVEABLE);
}

```

### Screen shot of above example





# Chapter 15

## Antialiasing

---

Lines are approximated by a series of pixels that must lie at display coordinates. They can therefore appear jagged, particularly lines which are nearly horizontal or nearly vertical. This jaggedness is called aliasing.

Antialiasing is the smoothing of lines and curves. It reduces the jagged, stair-step appearance of any line that is not exactly horizontal or vertical.  $\mu$ C/GUI supports different antialiasing qualities, antialiased fonts and high-resolution coordinates.

Support for antialiasing is a separate software item and is not included in the  $\mu$ C/GUI basic package. The software for antialiasing is located in the subdirectory `GUI\Anti-Alias`.

## 15.1 Introduction

Antialiasing smoothes curves and diagonal lines by "blending" the background color with that of the foreground. The higher the number of shades used between background and foreground colors, the better the antialiasing result (and the longer the computation time).

### Quality of antialiasing



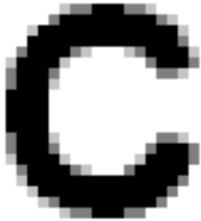

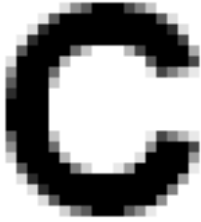

The quality of antialiasing is set by the routine `GUI_AA_SetFactor`, explained later in the chapter. For an idea of the relationship between the antialiasing factor and the corresponding result, take a look at the image pictured.

The first line is drawn without antialiasing (factor 1). The second line is drawn antialiased using factor 2. This means that the number of shades from foreground to background is  $2 \times 2 = 4$ . The next line is drawn with an antialiasing factor of 3, so there are  $3 \times 3 = 9$  shades, and so on. Factor 4 should be sufficient for most applications. Increasing the antialiasing factor further does not improve the result dramatically, but increases the calculation time.



## Antialiased Fonts

Two types of antialiased fonts, low-quality (2bpp) and high-quality (4bpp), are supported. The routines needed to display these fonts are automatically linked when using them. The following table shows the effect on drawing the character "C" without antialiasing and with both types of antialiased fonts:

Font type	Black on white	White on black
<b>Standard (no antialiasing)</b> 1 bpp 2 shades		
<b>Low-quality (antialiased)</b> 2 bpp 4 shades		
<b>High-quality (antialiased)</b> 4 bpp 16 shades		

Antialiased fonts can be created with the  $\mu$ C/GUI font converter. The general purpose of using antialiased fonts is to improve the appearance of text. While the effect of using high-quality antialiasing will be more visually pleasing than low-quality, computation time and memory consumption will increase proportionally. Low-quality (2bpp) fonts require twice the memory of non-antialiased (1bpp) fonts; high-quality (4bpp) fonts require four times the memory.

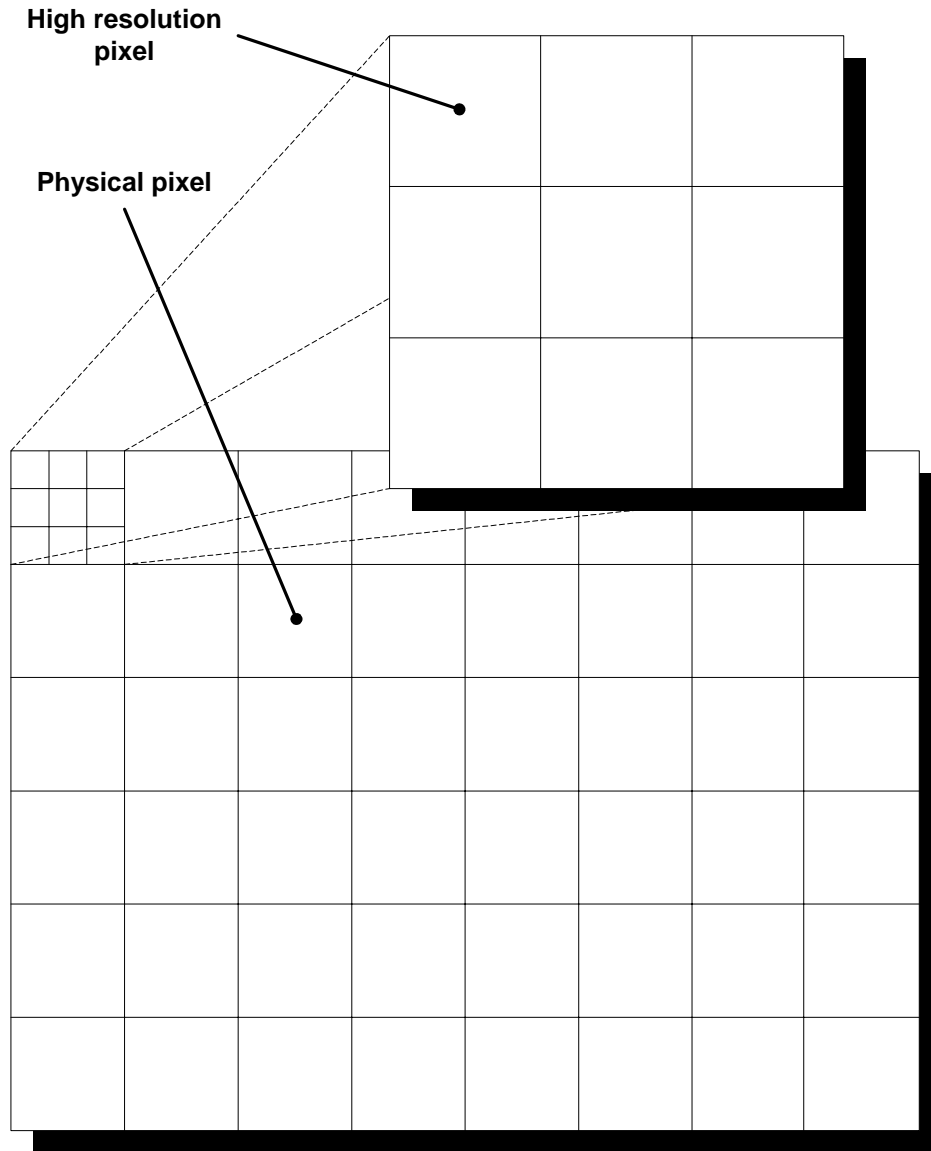
## High-resolution coordinates

When drawing items using antialiasing, the same coordinates are used as for regular (non-antialiasing) drawing routines. This is the default mode. You do not need to consider the antialiasing factor in the function arguments. For example, to draw an antialiased line from (50, 100) to (100, 50) you would write:

```
GUI_AA_DrawLine(50, 100, 100, 50);
```

The high-resolution feature of  $\mu$ C/GUI lets you use the virtual space determined by the antialiasing factor and your display size. High-resolution coordinates must be enabled with the routine `GUI_AA_EnableHiRes`, and may be disabled with `GUI_AA_DisableHiRes`. Both functions are explained later in the chapter. The advan-

tage of using high-resolution coordinates is that items can be placed not only at physical positions of your display but also "between" them. The virtual space of a high-resolution pixel is illustrated below based on an antialiasing factor of 3:



To draw a line from pixel (50, 100) to (100, 50) in high-resolution mode with anti-aliasing factor 3, you would write:

```
GUI_AA_DrawLine(150, 300, 300, 150);
```

For sample programs using the high-resolution feature, see the examples at the end of this chapter.

## 15.2 Antialiasing API

The table below lists the available routines in the antialiasing package, in alphabetical order within their respective categories. Detailed descriptions of the routines can be found in the sections that follow.

Routine	Explanation
Control functions	
<a href="#">GUI_AA_DisableHiRes()</a>	Disable high-resolution coordinates.
<a href="#">GUI_AA_EnableHiRes()</a>	Enable high-resolution coordinates.
<a href="#">GUI_AA_GetFactor()</a>	Return the current antialiasing factor.
<a href="#">GUI_AA_SetFactor()</a>	Set the current antialiasing factor.
Drawing functions	
<a href="#">GUI_AA_DrawArc()</a>	Draw an antialiased arc.
<a href="#">GUI_AA_DrawLine()</a>	Draw an antialiased line.
<a href="#">GUI_AA_DrawPolyOutline()</a>	Draw the outline of an antialiased polygon.
<a href="#">GUI_AA_FillCircle()</a>	Draw an antialiased circle.
<a href="#">GUI_AA_FillPolygon()</a>	Draw a filled and antialiased polygon.

## 15.3 Control functions

### GUI\_AA\_DisableHiRes()

#### Description

Disables high-resolution coordinates.

#### Prototype

```
void GUI_AA_DisableHiRes(void);
```

#### Additional Information

High-resolution coordinates are disabled by default.

### GUI\_AA\_EnableHiRes()

#### Description

Enables high-resolution coordinates.

#### Prototype

```
void GUI_AA_EnableHiRes(void);
```

### GUI\_AA\_GetFactor()

#### Description

Returns the current antialiasing quality factor.

#### Prototype

```
int GUI_AA_GetFactor(void);
```

**Return value**

The current antialiasing factor.

**GUI\_AA\_SetFactor()****Description**

Sets the antialiasing quality factor.

**Prototype**

```
void GUI_AA_SetFactor(int Factor);
```

Parameter	Meaning
<code>Factor</code>	The new antialiasing factor. Minimum: 1 (will result in no antialiasing) Maximum: 6

**Additional Information**

Setting the parameter `Factor` to 1, though permitted, will effectively disable antialiasing and result in a standard font.

We recommend an antialiasing quality factor of 2-4. The default factor is 3.

## 15.4 Drawing functions

**GUI\_AA\_DrawArc()****Description**

Displays an antialiased arc at a specified position in the current window, using the current pen size and the current pen shape.

**Prototype**

```
void GUI_AA_DrawArc(int x0, int y0, int rx, int ry, int a0, int a1);
```

Parameter	Meaning
<code>x0</code>	Horizontal position of the center.
<code>y0</code>	Vertical position of the center.
<code>rx</code>	Horizontal radius.
<code>ry</code>	Vertical radius.
<code>a0</code>	Starting angle (degrees).
<code>a1</code>	Ending angle (degrees).

**Limitations**

Currently the `ry` parameter is not available. The `rx` parameter is used instead.

**Additional Information**

If working in high-resolution mode, position and radius must be in high-resolution coordinates. Otherwise they must be specified in pixels.

## GUI\_AA\_DrawLine()

### Description

Displays an antialiased line at a specified position in the current window, using the current pen size and the current pen shape.

### Prototype

```
void GUI_AA_DrawLine(int x0, int y0, int x1, int y1);
```

Parameter	Meaning
<code>x0</code>	X-starting position.
<code>y0</code>	Y-starting position.
<code>x1</code>	X-end position.
<code>y1</code>	Y-end position.

### Additional Information

If working in high-resolution mode, the coordinates must be in high-resolution coordinates. Otherwise they must be specified in pixels.

## GUI\_AA\_DrawPolyOutline()

### Description

Displays the outline of an antialiased polygon defined by a list of points, at a specified position in the current window and with a specified thickness.

### Prototype

```
void GUI_AA_DrawPolyOutline(const GUI_POINT* pPoint,
                           int NumPoints,
                           int Thickness,
                           int x,
                           int y)
```

Parameter	Meaning
<code>pPoint</code>	Pointer to the polygon to display.
<code>NumPoints</code>	Number of points specified in the list of points.
<code>Thickness</code>	Thickness of the outline.
<code>x</code>	X-position of origin.
<code>y</code>	Y-position of origin .

### Additional Information

The polyline drawn is automatically closed by connecting the endpoint to the starting point. The starting point must not be specified a second time as an endpoint.

If working in high-resolution mode, the coordinates must be in high-resolution coordinates. Otherwise they must be specified in pixels.

### Example

```
#define countof(Array) (sizeof(Array) / sizeof(Array[0]))

static GUI_POINT aPoints[] = {
    { 0, 0 },
    { 15, 30 },
    { 0, 20 },
    { -15, 30 }
```

```
};  
  
void Sample(void) {  
    GUI_AA_DrawPolyOutline(aPoints, countof(aPoints), 3, 150, 40);  
}
```

Screen shot for preceeding example



GUI\_AA\_FillCircle()

Description

Displays a filled, antialiased circle at a specified position in the current window.

Prototype

```
void GUI_AA_FillCircle(int x0, int y0, int r);
```

Parameter	Meaning
x0	X-position of the center of the circle in pixels of the client window.
y0	Y-position of the center of the circle in pixels of the client window.
r	Radius of the circle (half of the diameter). Minimum: 0 (will result in a point). Maximum: 180.

Additional Information

If working in high-resolution mode, the coordinates must be in high-resolution coordinates. Otherwise they must be specified in pixels.

GUI\_AA\_FillPolygon()

Description

Fills an antialiased polygon defined by a list of points, at a specified position in the current window.

Prototype

```
void GUI_AA_FillPolygon(const GUI_POINT* pPoint,  
                        int NumPoints,  
                        int x,  
                        int y)
```

Parameter	Meaning
pPoint	Pointer to the polygon to display.
NumPoints	Number of points specified in the list of points.
x	X-position of origin.
y	Y-position of origin.

Additional Information

The polyline drawn is automatically closed by connecting the endpoint to the starting point. The starting point must not be specified a second time as an endpoint.

If working in high-resolution mode, the coordinates must be in high-resolution coordinates. Otherwise they must be specified in pixels.

## 15.5 Examples

### Different antialiasing factors

The following example creates diagonal lines with and without antialiasing. The source code can be found under `Sample\Misc\AntialiasedLines.c`.

```

/*****
 *
 *          Micrium Inc.
 *      Empowering embedded systems
 *
 *          µC/GUI sample code
 *
 *****/

-----
File      : AntialiasedLines.c
Purpose   : Shows lines with different antialiasing qualities
-----
*/

#include "GUI.H"

/*****
 *
 *      Show lines with different antialiasing qualities
 *
 *****/

static void DemoAntialiasing(void) {
    int i, x1, x2;
    int y = 2;
    /* Set drawing attributes */
    GUI_SetColor(GUI_BLACK);
    GUI_SetBkColor(GUI_WHITE);
    GUI_SetPenShape(GUI_PS_FLAT);
    GUI_Clear();
    x1 = 10; x2 = 90;
    /* Draw lines without antialiasing */
    GUI_DispStringHCenterAt("\nNormal", (x1 + x2) / 2, 10);
    for (i = 1; i < 12; i++) {
        GUI_SetPenSize(i);
        GUI_DrawLine(x1, 40 + i * 15, x2, 40 + i * 15 + y);
    }
    x1 = 110; x2 = 190;
    /* Draw lines with antialiasing quality faktor 2 */
    GUI_AA_SetFactor(2);
    GUI_DispStringHCenterAt("Antialiased\n\nusing factor 2", (x1 + x2) / 2, 10);
    for (i = 1; i < 12; i++) {
        GUI_SetPenSize(i);
        GUI_AA_DrawLine(x1, 40 + i * 15, x2, 40 + i * 15 + y);
    }
    x1 = 210; x2 = 290;
    /* Draw lines with antialiasing quality faktor 6 */
    GUI_AA_SetFactor(6);
    GUI_DispStringHCenterAt("Antialiased\n\nusing factor 6", (x1 + x2) / 2, 10);
    for (i = 1; i < 12; i++) {
        GUI_SetPenSize(i);
        GUI_AA_DrawLine(x1, 40 + i * 15, x2, 40 + i * 15 + y);
    }
}

/*****
 *
 *      main
 *
 *****/

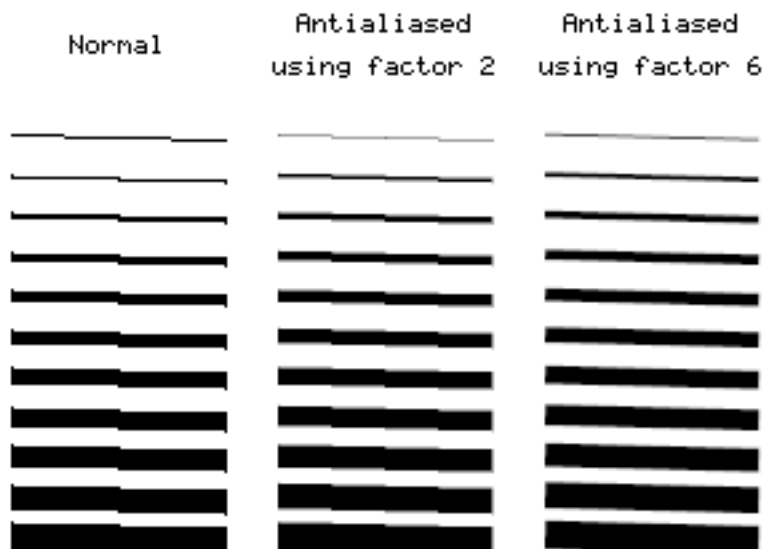
```

```

void main(void) {
    GUI_Init();
    DemoAntialiasing();
    while(1)
        GUI_Delay(100);
}

```

### Screen shot for preceeding example



### Lines placed on high-resolution coordinates

This example shows antialiased lines placed on high-resolution coordinates. It can be found under Sample\Misc\HiResPixel.c.

```

/*****
 *                               Micrium Inc.                               *
 *                               Empowering embedded systems               *
 *                               uC/GUI sample code                       *
 *                               *****/
/*****

-----
File       : HiResPixel.c
Purpose    : Demonstrates high resolution pixels
-----
*/

#include "GUI.H"

/*****
 *                               Show lines placed on high resolution pixels
 *                               *****/
*/

void ShowHiResPixel(void) {
    int i, Factor = 5;
    GUI_SetBkColor(GUI_WHITE);
    GUI_SetColor(GUI_BLACK);
    GUI_Clear();
    GUI_SetPenSize(2);
    GUI_SetPenShape(GUI_PS_FLAT);
    GUI_AA_EnableHiRes();          /* Enable high resolution */
}

```

```

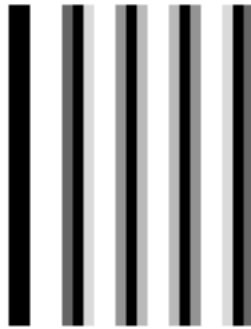
GUI_AA_SetFactor(Factor); /* Set quality factor */
/* Drawing lines using the virtual resolution */
for (i = 0; i < Factor; i++) {
    int x = (i + 1) * 5 * Factor + i - 1;
    GUI_AA_DrawLine(x, 50, x, 199);
}
}

/*****
*
*          main
*
*****/

void main(void) {
    GUI_Init();
    ShowHiResPixel();
    while(1);
}

```

### Magnified Screen shot for preceeding example



### Moving pointer using high-resolution antialiasing

This example illustrates the use of high-resolution antialiasing by drawing a rotating pointer that turns 0.1 degrees with each step. There is no screen shot of this example because the effects of high-resolution antialiasing are only visible in the movement of the pointers. Without high-resolution antialiasing the pointer appears to make short "jumps", whereas in high-resolution mode there is no apparent jumping.

The example can be found under `Sample\Misc\HiResAntialiasing.c`.

```

/*****
*          Micrium Inc.
*      Empowering embedded systems
*
*          µC/GUI sample code
*
*****/

-----
File      : HiResAntialiasing.c
Purpose   : Demonstrates high resolution antialiasing
-----
*/

#include "GUI.H"

/*****
*
*          Data

```

```

*
*****
*/

#define countof(Obj) (sizeof(Obj)/sizeof(Obj[0]))

static const GUI_POINT aPointer[] = {
    { 0, 3},
    { 85, 1},
    { 90, 0},
    { 85, -1},
    { 0, -3}
};

static GUI_POINT aPointerHiRes[countof(aPointer)];

typedef struct {
    GUI_AUTODEV_INFO AutoInfo;
    GUI_POINT aPoints[countof(aPointer)];
    int Factor;
} PARAM;

/*****
*
*           Drawing routines
*
*****/

static void DrawHiRes(void * p) {
    PARAM * pParam = (PARAM *)p;
    if (pParam->AutoInfo.DrawFixed) {
        GUI_ClearRect(0, 0, 99, 99);
    }
    GUI_AA_FillPolygon(pParam->aPoints,
                      countof(aPointer),
                      5 * pParam->Factor,
                      95 * pParam->Factor);
}

static void Draw(void * p) {
    PARAM * pParam = (PARAM *)p;
    if (pParam->AutoInfo.DrawFixed) {
        GUI_ClearRect(100, 0, 199, 99);
    }
    GUI_AA_FillPolygon(pParam->aPoints, countof(aPointer), 105, 95);
}

/*****
*
*   Demonstrate high resolution by drawing rotating pointers
*
*****/

static void ShowHiresAntialiasing(void) {
    int i;
    GUI_AUTODEV aAuto[2];
    PARAM Param;
    Param.Factor = 3;
    GUI_DispStringHCenterAt("Using\nhigh\nresolution\nmode", 50, 120);
    GUI_DispStringHCenterAt("Not using\nhigh\nresolution\nmode", 150, 120);
    /* Create GUI_AUTODEV objects */
    for (i = 0; i < countof(aAuto); i++) {
        GUI_MEMDEV_CreateAuto(&aAuto[i]);
    }
    /* Calculate pointer for high resolution */
    for (i = 0; i < countof(aPointer); i++) {
        aPointerHiRes[i].x = aPointer[i].x * Param.Factor;
        aPointerHiRes[i].y = aPointer[i].y * Param.Factor;
    }
    GUI_AA_SetFactor(Param.Factor); /* Set antialiasing factor */
    while(1) {

```

```

for (i = 0; i < 1800; i++) {
    float Angle = (i >= 900) ? 1800 - i : i;
    Angle *= 3.1415926f / 1800;
    /* Draw pointer with high resolution */
    GUI_AA_EnableHiRes();
    GUI_RotatePolygon(Param.aPoints, aPointerHiRes, countof(aPointer), Angle);
    GUI_MEMDEV_DrawAuto(&aAuto[0], &Param.AutoInfo, DrawHiRes, &Param);
    /* Draw pointer without high resolution */
    GUI_AA_DisableHiRes();
    GUI_RotatePolygon(Param.aPoints, aPointer, countof(aPointer), Angle);
    GUI_MEMDEV_DrawAuto(&aAuto[1], &Param.AutoInfo, Draw, &Param);
    #ifdef WIN32
        GUI_Delay(2);
    #endif
}
}
}

/*****
*
*           main
*
*****/

void main(void) {
    GUI_Init();
    ShowHiresAntialiasing();
}

```

# Chapter 16

## Unicode

---

The Unicode code standard is a 16-bit character encoding scheme. All of the characters available worldwide are in a single 16-bit character set (which works worldwide). The Unicode standard is defined by the Unicode consortium.

µC/GUI can display individual characters or strings in Unicode, although it is most common to simply use mixed strings, which can have any number of Unicode sequences within one ASCII string.

## 16.1 Displaying Unicode characters

### Unicode characters

The character output routine used by  $\mu$ C/GUI (`GUI_DispChar`) does always take an unsigned 16-bit value (U16) and has the basic ability to display a character defined by Unicode. It simply requires a font which contains the character you want to display.

### Displaying Unicode strings

The routine used for string output in  $\mu$ C/GUI is usually `GUI_DispString`. Since `GUI_DispString` uses 8-bit characters, you can not directly pass a Unicode string to it, as Unicode strings use 16-bit characters. There are two options available:

- Use `GUI_DispString_UC`, which accepts a 16-bit Unicode string, or
- Convert to an 8-bit string by embedding Unicode as 2-byte characters. This allows usage of the standard C string routines supplied with any compiler.

### Displaying Unicode mixed with ASCII code

This is the most recommended way to display Unicode. You do not have to use special functions to do so. There are only two macros for defining the beginning and the end of a Unicode sequence within one string.

Macro	Explanation
<code>GUI_UC_START</code>	Marks the start of a Unicode sequence.
<code>GUI_UC_END</code>	Marks the end of a Unicode sequence.

## 16.2 Unicode and double-byte conversions

### Why use the double-byte form?

First of all, there is no need to use the double-byte variant of Unicode strings. However, doing so may reduce memory consumption of text messages and can make it easier to deal with strings. This is true particularly if your compiler cannot handle 16-bit strings directly.

### How to form Unicode strings

The general idea is to be able to stay 100 percent compatible with existing software while still being able to use Unicode.

This means that strings are still byte = 8-bit arrays and that "normal" ASCII or Western European strings may be written as regular strings.

The beginning of a Unicode section is indicated by `GUI_UC_START`, which may be at any position in the string.

`GUI_UC_END` indicates the end of the Unicode section. A 0-character terminates the string as usual; if the last character was encoded in Unicode, `GUI_UC_END` may be omitted.

## Example

```

GUI_SetBkColor(GUI_RED);
GUI_SetColor(GUI_WHITE);
GUI_Clear();
GUI_SetFont(&GUI_Font16_1HK);
GUI_DispStringHCenterAt(
    "English mixed ... "
        GUI_UC_START /* Switch to UNICODE (double byte) */
        "\x30\x40" /* Japanese */
        "\x30\x45" /* Japanese */
        "\x30\x52" /* Japanese */
        "\x30\x51" /* Japanese */
        GUI_UC_END /* Back to single byte characters */
    " ..with Japanese",160,50);

```

Produces the following output:



## How does the double-byte form work?

All characters which do not have a code where either the high- or low-byte of the 16-bit character code is 0 will retain their 16-bit character code. Characters with high-byte = 0 code are moved to the area 0xe000-0xe0ff. For characters with low-byte = 0 code, the high-byte is used as the low-byte and the new high-byte is set to 0xe1. The resulting code is stored as 2 bytes, high-byte first.

## 16.3 Example

The following example defines a small font containing 6 characters: "A", "B", "C" and the Unicode characters 0x3060, 0x3061 and 0x3062. A mixed string is then drawn onto the display. The source code can be found under `Sample\Misc\Unicode.c`.

```

/*****
 *                               Micrium Inc.                               *
 *                               Empowering embedded systems                 *
 *                               uC/GUI sample code                         *
 *                               *****/
*****

-----
File       : Unicode.c
Purpose    : Example demonstrating UNICODE capabilities of uC/GUI
-----
*/

#include "GUI.H"

/*****
 *                               Definition of font containing UNICODE characters
 *                               *****/
*****

```

```

*/

/* Start of unicode area <Basic Latin> */
static const unsigned char acFontUC13_0041[ 13] = { /* code 0041 */
    _____,
    _____,
    _X_____,
    _X_____,
    _X_X_____,
    _X_X_____,
    _X_X_____,
    _XXXXX_____,
    _X_X_____,
    _X_X_____,
    _X_X_____,
    _X_X_____,
    _X_X_____,
    _____,
    _____};

static const unsigned char acFontUC13_0042[ 13] = { /* code 0042 */
    _____,
    _____,
    _XXXXX_____,
    _X_X_____,
    _X_X_____,
    _X_X_____,
    _XXXXX_____,
    _X_X_____,
    _X_X_____,
    _X_X_____,
    _X_X_____,
    _XXXXX_____,
    _____,
    _____};

static const unsigned char acFontUC13_0043[ 13] = { /* code 0043 */
    _____,
    _____,
    _XX_X_____,
    _X_XX_____,
    _X_X_____,
    _X_____,
    _X_____,
    _X_____,
    _X_____,
    _X_X_____,
    _XXX_____,
    _____,
    _____};

/* Start of unicode area <Hiragana> */
static const unsigned char acFontUC13_3060[ 26] = { /* code 3060 */
    _XX_,_X_X_,
    _X_,_X_X_,
    _X_XXX_,
    _XXXX_,
    _X_XX,XXX_,
    _X_,_X_,
    _X_X_,
    _X_,
    _X_X_,
    _X_X_,
    _X_X_,
    _X_XX,XXX_,
    _____,
    _____};

static const unsigned char acFontUC13_3061[ 26] = { /* code 3061 */
    _XX_,
    _X_,
    _X_XXXX,XX_,
    _XXXX_,
    _X_,
    _X_XXX,X_,
    _XXX_,_X_,
    _X_,_X_,
    _____,
    _____};

```

```

        _____, _X_____,
        _____, _X_____,
        _____, XXXXX, X_____,
        _____, _____,
        _____, _____};

static const unsigned char acFontUC13_3062[ 26] = { /* code 3062 */
    _____, X_X_____,
    _____, X_X_____,
    X_____XXXX, X_____,
    _____XXXX, _____,
    _____X_____, _____,
    _____X_XXX, X_____,
    _____XXX_____, X_____,
    _____X_____, X_____,
    _____, X_____,
    _____, X_____,
    _____XXXXX, X_____,
    _____, _____,
    _____, _____};

static const GUI_CHARINFO GUI_FontUC13_CharInfo[6] = {
    { 7, 7, 1, (void *)&acFontUC13_0041 } /* code 0041 */
    , { 7, 7, 1, (void *)&acFontUC13_0042 } /* code 0042 */
    , { 7, 7, 1, (void *)&acFontUC13_0043 } /* code 0043 */
    , { 14, 14, 2, (void *)&acFontUC13_3060 } /* code 3060 */
    , { 14, 14, 2, (void *)&acFontUC13_3061 } /* code 3061 */
    , { 14, 14, 2, (void *)&acFontUC13_3062 } /* code 3062 */
};

static const GUI_FONT_PROP GUI_FontUC13_Prop2 = {
    0x3060 /* first character */
    , 0x3062 /* last character */
    , &GUI_FontUC13_CharInfo[ 3] /* address of first character */
    , (void*)0 /* pointer to next GUI_FONT_PROP */
};

static const GUI_FONT_PROP GUI_FontUC13_Prop1 = {
    0x0041 /* first character */
    , 0x0043 /* last character */
    , &GUI_FontUC13_CharInfo[ 0] /* address of first character */
    , (void *)&GUI_FontUC13_Prop2 /* pointer to next GUI_FONT_PROP */
};

static const GUI_FONT GUI_FontUC13 = {
    GUI_FONTTYPE_PROP /* type of font */
    , 13 /* height of font */
    , 13 /* space of font y */
    , 1 /* magnification x */
    , 1 /* magnification y */
    , (void *)&GUI_FontUC13_Prop1
};

/*****
 *
 * Definition of string containing ASCII and UNICODE characters
 *
 *****/

static const char sUC_ASCII [] = {
    "ABC"GUI_UC_START"\x30\x60\x30\x61\x30\x62"GUI_UC_END"\x0"
};

/*****
 *
 * Demonstrates output of UNICODE characters
 *
 *****/

static void DemoUNICODE(void) {
    GUI_SetFont(&GUI_Font13HB_1);

```

```

GUI_DispStringHCenterAt("µC/GUI-sample: UNICODE characters", 160, 0);
/* Set ShiftJIS font */
GUI_SetFont(&GUI_FontUC13);
/* Display string */
GUI_DispStringHCenterAt(sUC_ASCII, 160, 40);
}

/*****
*
*          main
*
*****/

void main(void) {
    GUI_Init();
    DemoUNICODE();
    while(1)
        GUI_Delay(100);
}

```

### Screen shot for preceeding example

µC/GUI sample: UNICODE characters

ABCだぢぢ

# Chapter 17

## Shift-JIS Support

---

The most common Japanese encoding method is Shift-JIS. Shift-JIS encoding makes generous use of 8-bit characters, and the value of the first byte is used to distinguish single- and multiple-byte characters.

You need no special function calls to display a Shift JIS string. The main requirement is a font which contains the Shift JIS characters to be displayed.

## 17.1 Creating Shift-JIS fonts

The FontConverter can generate a Shift JIS font for  $\mu$ C/GUI from any Windows font. When using a Shift JIS font, the functions used to display Shift JIS characters are linked automatically with the library.

For detailed information on how to create Shift-JIS fonts, please contact Micrium (info@micrium.com). A separate FontConverter documentation describes all you need for an efficient way of implementing Shift JIS in your  $\mu$ C/GUI projects.

## 17.2 Example

The following example defines a small font containing 6 characters: "A", "B", "C" and the Shift-JIS characters 0x8350 (KATAKANA LETTER KE), 0x8351 (KATAKANA LETTER GE) and 0x8352 (KATAKANA LETTER KO). A mixed string is then drawn onto the display. The example can be found under Sample\Misc\ShiftJIS.c.

```

/*****
*                                     *
*                               Micrium Inc.                               *
*                         Empowering embedded systems                      *
*                                     *
*                                $\mu$ C/GUI sample code                          *
*                                     *
*****/

-----
File      : ShiftJIS.c
Purpose   : Example demonstrating ShiftJIS capabilities of  $\mu$ C/GUI
-----
*/

#include "gui.h"

/*****
*                                     *
*                         Definition of ShiftJIS font                      *
*                                     *
*****/

/* LATIN CAPITAL LETTER A */
static const unsigned char acFontSJIS13_0041[ 13] = { /* code 0041 */
    _____,
    _____,
    _X_____,
    _X_____,
    _X_X____,
    _X_X____,
    _X_X____,
    _XXXXX_,
    _X_X____,
    _X_X____,
    _X_X____,
    _X_X____,
    _X_X____,
    _____};

/* LATIN CAPITAL LETTER B */
static const unsigned char acFontSJIS13_0042[ 13] = { /* code 0042 */
    _____,
    _____,
    _XXXXX_,
    _X_X____,
    _X_X____,
    _X_X____,
    _____};

```

```

XXXX_,
X_X_,
X_X_,
X_X_,
XXXXX_,
_,
_};
/* LATIN CAPITAL LETTER C */
static const unsigned char acFontSJIS13_0043[ 13] = { /* code 0043 */
_,
XX_X_,
X_XX_,
X_X_,
X_,
X_,
X_,
X_,
X_X_,
XXX_,
_,
_};
/* KATAKANA LETTER KE */
static const unsigned char acFontSJIS13_8350[ 26] = { /* code 8350 */
XX_,
X_,
X_,
XXXXX,XXXX_,
X_X_,
X_X_,
X_X_,
X_,
X_,
X_,
XX_,
_,
_};
/* KATAKANA LETTER GE */
static const unsigned char acFontSJIS13_8351[ 26] = { /* code 8351 */
XX_,X_X_,
X_,X_X_,
X_,
XXXXXX,XXX_,
X_X_,
X_X_,
X_X_,
X_,
X_,
X_,
XX_,
_,
_};
/* KATAKANA LETTER KO */
static const unsigned char acFontSJIS13_8352[ 26] = { /* code 8352 */
_,
_,
XXXXXX,XX_,
_,X_,
_,X_,
_,X_,
_,X_,
_,X_,
_,X_,
XXXXXX,XXXX_,
_,
_,
_,
_};
static const GUI_CHARINFO GUI_FontSJIS13_CharInfo[6] = {

```

```

    { 7, 7, 1, (void *)&acFontSJIS13_0041 } /* code 0041 */
    , { 7, 7, 1, (void *)&acFontSJIS13_0042 } /* code 0042 */
    , { 7, 7, 1, (void *)&acFontSJIS13_0043 } /* code 0043 */
    , { 14, 14, 2, (void *)&acFontSJIS13_8350 } /* code 8350 */
    , { 14, 14, 2, (void *)&acFontSJIS13_8351 } /* code 8351 */
    , { 14, 14, 2, (void *)&acFontSJIS13_8352 } /* code 8352 */
};

static const GUI_FONT_PROP GUI_FontSJIS13_Prop2 = {
    0x8350 /* first character */
    ,0x8352 /* last character */
    ,&GUI_FontSJIS13_CharInfo[ 3] /* address of first character */
    ,(void*)0 /* pointer to next GUI_FONT_PROP */
};

static const GUI_FONT_PROP GUI_FontSJIS13_Prop1 = {
    0x0041 /* first character */
    ,0x0043 /* last character */
    ,&GUI_FontSJIS13_CharInfo[ 0] /* address of first character */
    ,(void *)&GUI_FontSJIS13_Prop2 /* pointer to next GUI_FONT_PROP */
};

static const GUI_FONT GUI_FontSJIS13 = {
    GUI_FONTTYPE_PROP_SJIS /* type of font */
    ,13 /* height of font */
    ,13 /* space of font y */
    ,1 /* magnification x */
    ,1 /* magnification y */
    ,(void *)&GUI_FontSJIS13_Prop1
};

/*****
 *
 * Definition of string containing ASCII and ShiftJIS characters
 *
 *****/

static const char aSJIS[] = {
    "ABC\x83\x50\x83\x51\x83\x52\x0"
};

/*****
 *
 * Demonstrates output of ShiftJIS characters
 *
 *****/

void DemoShiftJIS(void) {
    GUI_SetFont(&GUI_Font13HB_1);
    GUI_DispStringHCenterAt("µC/GUI-sample: ShiftJIS characters", 160, 0);
    /* Set ShiftJIS font */
    GUI_SetFont(&GUI_FontSJIS13);
    /* Display string */
    GUI_DispStringHCenterAt(aSJIS, 160, 40);
}

/*****
 *
 * main
 *
 *****/

void main(void) {
    GUI_Init();
    DemoShiftJIS();
    while(1)
        GUI_Delay(100);
}

```

**Screen shot for preceeding example**

**μC/GUI sample: UNICODE characters**

ABCだちち



# Chapter 18

## Input Devices

---

μC/GUI provides touch-screen, mouse, and keyboard support. The basic μC/GUI package includes a driver for analog touch-screens and a PS2 mouse driver, although other types of touch-panel and mouse devices can also be used with the appropriate drivers. Any type of keyboard driver is compatible with μC/GUI. The software for input devices is located in the subdirectory `GUI\Core`.

## 18.1 Pointer input devices

Pointer input devices include the mouse and the touch-screen. They share a group of common pointer input device (PID) routines so as to enable simultaneous mouse and touch-screen use. These routines are typically called automatically by the window manager, which reacts accordingly to update the display. If the window manager is not used, your application is responsible for calling the PID routines.

### Data structure

The structure of type `GUI_PID_STATE` referenced by the parameter `pState` is filled by the routine with the current values. The structure is defined as follows:

```
typedef struct {
    int x,y;
    unsigned char Pressed;
} GUI_PID_STATE;
```

### Pointer input device API

The table below lists the pointer input device routines in alphabetical order. Detailed descriptions follow.

Routine	Explanation
<code>GUI_PID_GetState()</code>	Return the current state of the PID.
<code>GUI_PID_StoreState()</code>	Store the current state of the PID.

#### GUI\_PID\_GetState()

##### Description

Returns the current state of the pointer input device.

##### Prototype

```
void GUI_PID_GetState(const GUI_PID_STATE *pState);
```

Parameter	Meaning
<code>pState</code>	Pointer to a structure of type <code>GUI_PID_STATE</code> .

##### Return value

1 if input device is currently pressed; 0 if not pressed.

#### GUI\_PID\_StoreState()

##### Description

Stores the current state of the pointer input device.

##### Prototype

```
int GUI_PID_StoreState(GUI_PID_STATE *pState);
```

Parameter	Meaning
<code>pState</code>	Pointer to a structure of type <code>GUI_PID_STATE</code> .

## 18.1.1 Mouse driver input

Mouse support consists of two "layers": a generic layer and a mouse driver layer. Generic routines refer to those functions which always exist, no matter what type of mouse driver you use. The available mouse driver routines, on the other hand, will call the appropriate generic routines as necessary, and may only be used with the PS2 mouse driver supplied with  $\mu$ C/GUI. If you write your own driver, it is responsible for calling the generic routines.

The generic mouse routines will in turn call the corresponding PID routines.

### Generic mouse API

The table below lists the generic mouse routines in alphabetical order. These functions may be used with any type of mouse driver. Detailed descriptions follow.

Routine	Explanation
<a href="#">GUI_MOUSE_GetState()</a>	Return the current state of the mouse.
<a href="#">GUI_MOUSE_StoreState()</a>	Store the current state of the mouse.

### GUI\_MOUSE\_GetState()

#### Description

Returns the current state of the mouse.

#### Prototype

```
int GUI_MOUSE_GetState(GUI_PID_STATE *pState);
```

Parameter	Meaning
<a href="#">pState</a>	Pointer to a structure of type GUI_PID_STATE.

#### Return value

1 if mouse is currently pressed; 0 if not pressed.

#### Additional information

This function will call `GUI_PID_GetState()`.

### GUI\_MOUSE\_StoreState()

#### Description

Stores the current state of the mouse.

#### Prototype

```
void GUI_MOUSE_StoreState(const GUI_PID_STATE *pState);
```

Parameter	Meaning
<a href="#">pState</a>	Pointer to a structure of type GUI_PID_STATE.

#### Additional information

This function will call `GUI_PID_StoreState()`.

## PS2 mouse driver API

The table below lists the available mouse driver routines in alphabetical order. These functions only apply if you are using the PS2 mouse driver included with  $\mu$ C/GUI.

Routine	Explanation
<code>GUI_MOUSE_DRIVER_PS2_Init()</code>	Initialize the mouse driver.
<code>GUI_MOUSE_DRIVER_PS2_OnRx()</code>	Called from receive interrupt routines.

### GUI\_MOUSE\_DRIVER\_PS2\_Init()

#### Description

Initializes the mouse driver.

#### Prototype

```
void GUI_MOUSE_DRIVER_PS2_Init(void);
```

### GUI\_MOUSE\_DRIVER\_PS2\_OnRx()

#### Description

Must be called from receive interrupt routines.

#### Prototype

```
void GUI_MOUSE_DRIVER_PS2_OnRx(unsigned char Data);
```

Parameter	Meaning
<code>Data</code>	Byte of data received by ISR.

#### Additional information

The PS2 mouse driver is a serial driver, meaning it receives 1 byte at a time. You need to ensure that this function is called from your receive interrupt routine every time a byte (1 character) is received.

## 18.1.2 Touch-screen driver input and configuration

Touch-screen support also consists of a generic layer and a driver layer. Generic routines can be used with any type of driver (analog, digital, etc.). The available driver routines will call the appropriate generic routines as necessary and may only be used with the analog touch-screen driver supplied with  $\mu$ C/GUI. As with mouse support, if you write your own driver, it is responsible for calling the generic routines.

The generic mouse routines will call the corresponding PID routines.

The driver layer also includes a configuration module which may require modifications.

## Generic touch-screen API

The table below lists the generic touch-screen routines in alphabetical order. These functions may be used with any type of touch-screen driver. Detailed descriptions follow.

Routine	Explanation
<a href="#">GUI_TOUCH_GetState()</a>	Return the current state of the touch-screen.
<a href="#">GUI_TOUCH_StoreState()</a>	Store the current state of the touch-screen.

### GUI\_TOUCH\_GetState()

#### Description

Returns the current state of the touch-screen.

#### Prototype

```
int GUI_TOUCH_GetState(GUI_PID_STATE *pState);
```

Parameter	Meaning
<a href="#">pState</a>	Pointer to a structure of type GUI_PID_STATE.

#### Return value

1 if touch-screen is currently pressed; 0 if not pressed.

### GUI\_TOUCH\_StoreState()

#### Description

Stores the current state of the touch-screen.

#### Prototype

```
void GUI_TOUCH_StoreState(int x int y);
```

Parameter	Meaning
<a href="#">x</a>	X-position.
<a href="#">y</a>	Y-position.

#### Additional information

This function will call `GUI_PID_StoreState()`.

## Driver API for analog touch-screens

The  $\mu$ C/GUI touch-screen driver handles analog input (from an 8-bit or better A/D converter), debouncing and calibration of the touch-screen.

The touch-screen driver continuously monitors and updates the touch-panel through the use of the function `GUI_TOUCH_Exec()`, which calls the appropriate generic touch-screen API routines when it recognizes that an action has been performed or something has changed.

The table below lists the available analog touch-screen driver routines in alphabetical order. These functions only apply if you are using the driver included with  $\mu$ C/GUI..

Routine	Explanation
<a href="#">GUI_TOUCH_Calibrate()</a>	Changes the calibration.
<a href="#">GUI_TOUCH_Exec()</a>	Activates the measurement of the X- and Y-axes; needs to be called about 100 times/second.
<a href="#">GUI_TOUCH_SetDefaultCalibration()</a>	Restores the default calibration.

## TOUCH\_X routines

The following four hardware-dependent functions need to be added to your project if you use the driver supplied with  $\mu$ C/GUI, as they are called by [GUI\\_TOUCH\\_Exec\(\)](#) when polling the touch-panel. A suggested place is in the file [GUI\\_X.c](#). These functions are as follows:

Routine	Explanation
<a href="#">TOUCH_X_ActivateX()</a>	Prepares measurement for Y-axis.
<a href="#">TOUCH_X_ActivateY()</a>	Prepares measurement for X-axis.
<a href="#">TOUCH_X_MeasureX()</a>	Returns the X-result of the A/D converter.
<a href="#">TOUCH_X_MeasureY()</a>	Returns the Y-result of the A/D converter.

## GUI\_TOUCH\_Calibrate()

### Description

Changes the calibration at run-time.

### Prototype

```
int GUI_TOUCH_Calibrate(int Coord, int Log0, int Log1, int Phys0, int
                        Phys1);
```

Parameter	Meaning
<a href="#">Coord</a>	0 for X-axis, 1 for Y-axis.
<a href="#">Log0</a>	Logical value 0 in pixels.
<a href="#">Log1</a>	Logical value 1 in pixels.
<a href="#">Phys0</a>	A/D converter value for Log0.
<a href="#">Phys1</a>	A/D converter value for Log1.

### Additional information

The function takes as parameters the axis to be calibrated, two logical values in pixels for this axis and two corresponding physical values of the A/D converter.

## GUI\_TOUCH\_Exec()

### Description

Polls the touch-screen by calling the [TOUCH\\_X](#) routines to activate the measurement of the X- and Y-axes. You must ensure that this function is called for about 100 times per second.

### Prototype

```
void GUI_TOUCH_Exec(void);
```

### Additional information

If you are using a real-time operating system, the easiest way to make sure this function is called is to create a separate task. When not using a multitask system, you can use an interrupt service routine to do the job.

## GUI\_TOUCH\_SetDefaultCalibration()

### Description

Resets the calibration to the values set as default in the configuration file.

### Prototype

```
void GUI_TOUCH_SetDefaultCalibration(void);
```

### Additional information

If no values are set in the configuration file, the calibration will be restored to the original default values.

## TOUCH\_X\_ActivateX(), TOUCH\_X\_ActivateY()

### Description

These routines are called from `GUI_TOUCH_Exec()` to activate the measurement of the X- and the Y-axes. `TOUCH_X_ActivateX()` switches on the measurement voltage to the X-axis; `TOUCH_X_ActivateY()` switches on the voltage to the Y-axis. Switching on the voltage in X means the value for the Y-axis can be measured and vice versa.

### Prototypes

```
void TOUCH_X_ActivateX(void);
void TOUCH_X_ActivateY(void);
```

## TOUCH\_X\_MeasureX(), TOUCH\_X\_MeasureY()

### Description

These routines are called from `GUI_TOUCH_Exec()` to return the measurement values from the A/D converter for the X- and the Y-axes.

### Prototypes

```
int TOUCH_X_MeasureX(void);
int TOUCH_X_MeasureY(void);
```

## Configuring the touch-screen module

There needs to exist a separate configuration file in your `config` folder named `GUITouchConf.h`. The following table shows all available `config` macros for the analog touch-screen driver included with  $\mu$ C/GUI:

Type	Macro	Default	Explanation
B	GUI_TOUCH_SWAP_XY	0	Set to 1 to swap the X- and the Y-axes.
B	GUI_TOUCH_MIRROR_X	0	Mirrors the X-axis.
B	GUI_TOUCH_MIRROR_Y	0	Mirrors the Y-axis.
N	GUI_TOUCH_AD_LEFT	30	Minimum value returned by the A/D converter.
N	GUI_TOUCH_AD_RIGHT	220	Maximum value returned by the A/D converter.
N	GUI_TOUCH_AD_TOP	30	Minimum value returned by the A/D converter.

Type	Macro	Default	Explanation
N	GUI_TOUCH_AD_BOTTOM	220	Maximum value returned by the A/D converter.
N	GUI_TOUCH_XSIZE	LCD_XSIZE	Horizontal area covered by touch-screen.
N	GUI_TOUCH_YSIZE	LCD_YSIZE	Vertical area covered by touch-screen.

## 18.2 Keyboard input

A keyboard input device uses ASCII character coding in order to be able to distinguish between characters. For example, there is only one "A" key on the keyboard, but an uppercase "A" and a lowercase "a" have different ASCII codes (0x41 and 0x61, respectively).

### μC/GUI predefined character codes

μC/GUI also defines character codes for other "virtual" keyboard operations. These codes are listed in the table below, and defined in an identifier table in `GUI.h`. A character code in μC/GUI can therefore be any extended ASCII character value or any of the following predefined μC/GUI values.

Predefined virtual key code	Description
GUI_KEY_BACKSPACE	Backspace key.
GUI_KEY_TAB	Tab key.
GUI_KEY_ENTER	Enter/return key.
GUI_KEY_LEFT	Left arrow key.
GUI_KEY_UP	Up arrow key.
GUI_KEY_RIGHT	Right arrow key.
GUI_KEY_DOWN	Down arrow key.
GUI_KEY_HOME	Home key (move to beginning of current line).
GUI_KEY_END	End key (move to end of current line).
GUI_KEY_SHIFT	Shift key.
GUI_KEY_CONTROL	Control key.
GUI_KEY_ESCAPE	Escape key.
GUI_KEY_INSERT	Insert key.
GUI_KEY_DELETE	Delete key.

### 18.2.1 Driver layer API

The keyboard driver layer handles keyboard messaging functions. These routines notify the window manager when specific keys (or combinations of keys) have been pressed or released.

The table below lists the driver-layer keyboard routines in alphabetical order. Detailed descriptions follow.

Routine	Explanation
<a href="#">GUI_StoreKeyMsg()</a>	Store a message in a specified key.
<a href="#">GUI_SendKeyMsg()</a>	Send a message to a specified key.

## GUI\_StoreKeyMsg()

### Description

Stores a status message in a specified key.

### Prototype

```
void GUI_StoreKeyMsg(int Key, int Pressed);
```

Parameter	Meaning
<a href="#">Key</a>	May be any extended ASCII character (between 0x20 and 0xFF) or any predefined $\mu$ C/GUI character code.
<a href="#">Pressed</a>	Key state (see table below).

Permitted values for parameter <a href="#">Pressed</a>	
1	Pressed state.
0	Released (unpressed) state.

## GUI\_SendKeyMsg()

### Description

Sends a status message to a specified key.

### Prototype

```
void GUI_SendKeyMsg(int Key, int Pressed);
```

Parameter	Meaning
<a href="#">Key</a>	May be any extended ASCII character (between 0x20 and 0xFF) or any predefined $\mu$ C/GUI character code.
<a href="#">Pressed</a>	Key state (see <a href="#">GUI_StoreKeyMsg()</a> ).

## 18.2.2 Application layer API

The table below lists the application-layer keyboard routines in alphabetical order. Detailed descriptions follow.

Routine	Explanation
<a href="#">GUI_ClearKeyBuffer()</a>	Clear the key buffer.
<a href="#">GUI_GetKey()</a>	Return the contents of the key buffer.
<a href="#">GUI_StoreKey()</a>	Store a key in the buffer.
<a href="#">GUI_WaitKey()</a>	Wait for a key to be pressed.

## GUI\_ClearKeyBuffer()

**Description**  
Clears the key buffer.

**Prototype**  
`void GUI_ClearKeyBuffer(void);`

## GUI\_GetKey()

**Description**  
Returns the current contents of the key buffer.

**Prototype**  
`int GUI_GetKey(void);`

**Return value**  
Codes of characters in key buffer; 0 if no keys in buffer.

## GUI\_StoreKey()

**Description**  
Stores a key in the buffer.

**Prototype**  
`void GUI_StoreKey(int Key);`

Parameter	Meaning
Key	May be any extended ASCII character (between 0x20 and 0xFF) or any predefined $\mu$ C/GUI character code.

**Additional information**  
This function is typically called by the driver and not by the application itself.

## GUI\_WaitKey()

**Description**  
Waits for a key to be pressed.

**Prototype**  
`int GUI_WaitKey(void);`

**Additional information**  
The application is "blocked", meaning it will not return until a key is pressed.

# Chapter 19

## Time-Related Functions

---

Some widgets, as well as our demonstration code, require time-related functions. The other parts of the  $\mu$ C/GUI graphic library do not require a time base. The demonstration code makes heavy use of the routine `GUI_Delay()`, which delays for a given period of time. A unit of time is referred to as a tick.

## Timing and execution API

The table below lists the available timing- and execution-related routines in alphabetical order. Detailed descriptions of the routines follow.

Routine	Explanation
<a href="#">GUI_Delay()</a>	Delay for a specified period of time.
<a href="#">GUI_Exec()</a>	Execute callback functions (all jobs).
<a href="#">GUI_Exec1()</a>	Execute one callback function (one job only).
<a href="#">GUI_GetTime()</a>	Return the current system time.

### GUI\_Delay()

#### Description

Delays for a specified period of time.

#### Prototype

```
void GUI_Delay(int Period);
```

Parameter	Explanation
<a href="#">Period</a>	Period in ticks until function should return.

#### Additionnal information

The time unit (tick) is usually milliseconds (depending on [GUI\\_X\\_](#) functions).

[GUI\\_Delay\(\)](#) only executes idle functions for the given period. If the window manager is used, the delay time is used for the updating of invalid windows (through execution of [WM\\_Exec\(\)](#)).

This function will call [GUI\\_X\\_Delay\(\)](#).

### GUI\_Exec()

#### Description

Executes callback functions (typically redrawing of windows).

#### Prototype

```
int GUI_Exec(void);
```

#### Return value

0 if there were no jobs performed.

1 if a job was performed.

#### Additionnal information

This function will automatically call [GUI\\_Exec1\(\)](#) repeatedly until it has completed all jobs -- essentially until a 0 value is returned.

Normally this function does not need to be called by the user application. It is called automatically by [GUI\\_Delay\(\)](#).

### GUI\_Exec1()

#### Description

Executes a callback function (one job only -- typically redrawing a window).

**Prototype**

```
int GUI_Exec1(void);
```

**Return value**

0 if there were no jobs performed.  
1 if a job was performed.

**Additional information**

This routine may be called repeatedly until 0 is returned, which means all jobs have been completed.

This function is called automatically by `GUI_Exec()`.

**GUI\_GetTime()****Description**

Returns the current system time.

**Prototype**

```
int GUI_GetTime(void);
```

**Return value**

The current system time in ticks.

**Additional information**

This function will call `GUI_X_GetTime()`.



# Chapter 20

## Low-Level Configuration

---

Before you can use  $\mu$ C/GUI on your target system, you need to configure the software for your application. Configuring means modifying the configuration (header) files which usually reside in the (sub)directory `Config`. We try to keep the configuration as simple as possible, but there are some configuration macros (in the file `LCD-Conf.h`) which you must modify in order for the system to work properly. These include:

- LCD macros, defining the size of the display as well as optional features (such as mirroring, etc.)
- LCD controller macros, defining how to access the controller you are using.

## 20.1 Available configuration macros

The following table shows the available macros used for low-level configuration of  $\mu$ C/GUI:

Type	Macro	Default	Explanation
General (required) configuration			
S	LCD_CONTROLLER	---	Select LCD controller.
N	LCD_BITSPERPIXEL	---	Specify bits per pixel.
S	LCD_FIXEDPALETTE	---	Specify fixed palette mode. Set to 0 for a user-defined color lookup table (then LCD_PHYSCOLORS must be defined).
N	LCD_XSIZE	---	Define horizontal resolution of LCD.
N	LCD_YSIZE	---	Define vertical resolution of LCD.
Initializing the controller			
F	LCD_INIT_CONTROLLER( )	---	Initialization sequence for the LCD controller(s). Not applicable with all controllers.
Display orientation			
B	LCD_MIRROR_X	0	Activate to mirror X-axis.
B	LCD_MIRROR_Y	0	Activate to mirror Y-axis.
B	LCD_SWAP_XY	0	Activate to swap X- and Y-axes. If set to 0, SEG lines refer to columns and COM lines refer to rows.
N	LCD_VXSIZE	LCD_XSIZE	Horizontal resolution of virtual display. Not applicable with all drivers.
N	LCD_VYSIZE	LCD_YSIZE	Vertical resolution of virtual display. Not applicable with all drivers.
N	LCD_XORG<n>	0	LCD controller <n>: leftmost (lowest) X-position.
N	LCD_YORG<n>	0	LCD controller <n>: topmost (lowest) Y-position.
Color configuration			
N	LCD_MAX_LOG_COLORS	256	Maximum number of logical colors that the driver can support in a bitmap.
A	LCD_PHYSCOLORS	---	Defines the contents of the color lookup table. Only required if LCD_FIXEDPALETTE is set to 0.
B	LCD_PHYSCOLORS_IN_RAM	0	Only relevant if physical colors are defined. Puts physical colors in RAM, making them modifiable at run time.
B	LCD_REVERSE	0	Activate to invert the display at compile time.
B	LCD_SWAP_RB	0	Activate to swap the red and blue components.
Magnifying the LCD			
N	LCD_XMAG<n>	1	Horizontal magnification factor of LCD.
N	LCD_YMAG<n>	1	Vertical magnification factor of LCD.
Simple bus interface configuration			
F	LCD_READ_A0(Result)	---	Read a byte from LCD controller with A-line low.
F	LCD_READ_A1(Result)	---	Read a byte from LCD controller with A-line high.
F	LCD_WRITE_A0(Byte)	---	Write a byte to LCD controller with A-line low.
F	LCD_WRITE_A1(Byte)	---	Write a byte to LCD controller with A-line high.
F	LCD_WRITE_M_A1	---	Write multiple bytes to LCD controller with A-line high.

Type	Macro	Default	Explanation
Full bus interface configuration			
F	LCD_READ_MEM( Index )	---	Read the contents of video memory of controller.
F	LCD_READ_REG( Index )	---	Read the contents of a configuration register of controller.
F	LCD_WRITE_MEM( Index, Data )	---	Write to video memory (display data RAM) of controller.
F	LCD_WRITE_REG( Index, Data )	---	Write to a configuration register of controller.
S	LCD_BUSWIDTH	16	Select bus-width (8/16) of LCD controller/CPU interface.
F	LCD_ENABLE_REG_ACCESS	---	Switch the M/R signal to register access. Not applicable with all controllers.
F	LCD_ENABLE_MEM_ACCESS	---	Switch the M/R signal to memory access. Not applicable with all controllers.
B	LCD_SWAP_BYTE_ORDER	0	Activate to invert the endian mode (swap the high and low bytes) between CPU and LCD controller when using a 16-bit bus interface.
LCD controller configuration: common/segment lines			
N	LCD_XORG<n>	0	LCD controller <n>: leftmost (lowest) X-position.
N	LCD_YORG<n>	0	LCD controller <n>: topmost (lowest) Y-position.
N	LCD_FIRSTSEG<n>	0	LCD controller <n>: first segment line used.
N	LCD_LASTSEG<n>	LCD_XSIZE-1	LCD controller <n>: last segment line used.
N	LCD_FIRSTCOM<n>	0	LCD controller <n>: first common line used.
N	LCD_LASTCOM<n>	LCD_YSIZE-1	LCD controller <n>: last common line used.
COM/SEG lookup tables			
A	LCD_LUT_COM	---	COM lookup table for controller.
A	LCD_LUT_SEG	---	SEG lookup table for controller.
Miscellaneous			
N	LCD_NUM_CONTROLLERS	1	Number of LCD controllers used.
B	LCD_CACHE	1	Deactivate to disable use of display data cache, which slows down the speed of the driver. Not applicable with all drivers.
B	LCD_USE_BITBLT	1	Deactivate to disable BitBLT engine. If set to 1, the driver will use all available hardware acceleration.
B	LCD_SUPPORT_CACHECONTROL	0	Activate to enable cache control functions of LCD_L0_ControlCache( ) driver API. Not applicable with all controllers.
N	LCD_TIMERINIT0	---	Timing value used by ISR for displaying pane 0 when using CPU as controller.
N	LCDTIMERINIT1	---	Timing value used by ISR for displaying pane 1 when using CPU as controller.
F	LCD_ON	---	Function replacement macro which switches the LCD on.
F	LCD_OFF	---	Function replacement macro which switches the LCD off.

## How to configure the LCD

We recommend the following procedure:

1. Make a copy of a configuration file of similar configuration. Several configuration samples for your particular LCD controller can be found in the folder Sam-

ple\LCDConf\xxx, where xxx is your LCD driver.

2. Configure the bus interface by defining the appropriate simple bus or full bus macros.
3. Define the size of your LCD (LCD\_XSIZE, LCD\_YSIZE).
4. Select the controller used in your system, as well as the appropriate bpp and the palette mode (LCD\_CONTROLLER, LCD\_BITSPERPIXEL, LCD\_FIXEDPALETTE).
5. Configure which common/segment lines are used, if necessary.
6. Test the system.
7. Reverse X/Y if necessary (LCD\_REVERSE); go back to step 6 in this case.
8. Mirror X/Y if necessary (LCD\_MIRROR\_X, LCD\_MIRROR\_Y); go back to step 6 in this case.
9. Check all the other configuration switches.
10. Erase unused sections of the configuration.

## 20.2 General (required) configuration

### LCD\_CONTROLLER

#### Description

Defines the LCD controller used.

#### Type

Selection switch

#### Additional information

The LCD controller used is designated by the appropriate number. Please refer to Chapter 22: "LCD Drivers" for more information about available options.

#### Example

Specifies an Epson SED1565 controller:

```
#define LCD_CONTROLLER 1565 /* Selects SED 1565 LCD-controller */
```

### LCD\_BITSPERPIXEL

#### Description

Specifies the number of bits per pixel.

#### Type

Numerical value

### LCD\_FIXEDPALETTE

#### Description

Specifies the fixed palette mode.

#### Type

Selection switch

#### Additional information

Set the value to 0 to use a color lookup table instead of a fixed palette mode. The macro LCD\_PHYSCOLORS must then be defined.

## LCD\_XSIZE; LCD\_YSIZE

### Description

Define the horizontal and vertical resolution (respectively) of the display used.

### Type

Numerical values

### Additional information

The values are logical sizes; X-direction specifies the direction which is used as the X-direction by all routines of the LCD driver.

Usually the X-size equals the number of segments.

## 20.3 Initializing the controller

### LCD\_INIT\_CONTROLLER()

#### Description

Initializes the LCD controller(s).

#### Type

Function replacement

#### Additional information

This macro must be user-defined to initialize some controllers. It is executed during the `LCD_L0_Init()` and `LCD_L0_Reinit()` routines of the driver. Please consult the data sheet of your controller for information on how to initialize your hardware.

#### Example

The sample below has been written for and tested with an Epson SED1565 controller using an internal power regulator.

```
#define LCD_INIT_CONTROLLER() \
LCD_WRITE_A0(0xe2); /* Internal reset                                */ /* \
LCD_WRITE_A0(0xae); /* Display on/off: off                          */ /* \
LCD_WRITE_A0(0xac); /* Power save start: static indicator off      */ /* \
LCD_WRITE_A0(0xa2); /* LCD bias select: 1/9                        */ /* \
LCD_WRITE_A0(0xa0); /* ADC select: normal                          */ /* \
LCD_WRITE_A0(0xc0); /* Common output mode: normal                 */ /* \
LCD_WRITE_A0(0x27); /* V5 voltage regulator: medium               */ /* \
LCD_WRITE_A0(0x81); /* Enter electronic volume mode                */ /* \
LCD_WRITE_A0(0x13); /* Electronic volume: medium                  */ /* \
LCD_WRITE_A0(0xad); /* Power save end: static indicator on         */ /* \
LCD_WRITE_A0(0x03); /* static indicator register set: on (constantly on) */ /* \
LCD_WRITE_A0(0x2F); /* Power control set: booster, regulator and follower off */ /* \
LCD_WRITE_A0(0x40); /* Display Start Line                          */ /* \
LCD_WRITE_A0(0xB0); /* Display Page Address 0                      */ /* \
LCD_WRITE_A0(0x10); /* Display Column Address MSB                 */ /* \
LCD_WRITE_A0(0x00); /* Display Column Address LSB                 */ /* \
LCD_WRITE_A0(0xaf); /* Display on/off: on                          */ /* \
LCD_WRITE_A0(0xe3); /* NOP                                         */ /*
```

## 20.4 Display orientation

### LCD\_MIRROR\_X

**Description**

Inverts the X-direction (horizontal) of the display.

**Type**

Binary switch

0: inactive, X not mirrored (default)

1: active, X mirrored

**Additional information**

If activated:  $X \rightarrow \text{LCD\_XSIZE} - 1 - X$ .

This macro, in combination with `LCD_MIRROR_Y` and `LCD_SWAP_XY`, can be used to support any orientation of the display. Before changing this configuration switch, make sure that `LCD_SWAP_XY` is set as required by your application.

### LCD\_MIRROR\_Y

**Description**

Inverts the Y-direction (vertical) of the display.

**Type**

Binary switch

0: inactive, Y not mirrored (default)

1: active, Y mirrored

**Additional information**

If activated:  $Y \rightarrow \text{LCD\_YSIZE} - 1 - Y$ .

This macro, in combination with `LCD_MIRROR_X` and `LCD_SWAP_XY`, can be used to support any orientation of the display. Before changing this configuration switch, make sure that `LCD_SWAP_XY` is set as required by your application.

### LCD\_SWAP\_XY

**Description**

Swaps the horizontal and vertical directions (orientation) of the display.

**Type**

Binary switch

0: inactive, X-Y not swapped (default)

1: active, X-Y swapped

**Additional information**

If set to 0 (not swapped), SEG lines refer to columns and COM lines refer to rows.

If activated:  $X \rightarrow Y$ .

When changing this switch, you will also have to swap the X-Y settings for the resolution of the display (using `LCD_XSIZE` and `LCD_YSIZE`).

### LCD\_VXSIZE; LCD\_VYSIZE

**Description**

Define the horizontal and vertical resolution (respectively) of the virtual display.

**Type**

Numerical values

**Additionnal information**

The values are logical sizes; X-direction specifies the direction which is used as X-direction by all routines of the LCD driver.

If a virtual display is not used, these values should be the same as the values for `LCD_XSIZE`, `LCD_YSIZE` (these are the default settings).

The virtual display feature requires hardware support and is not available with all drivers.

**LCD\_XORG<n>; LCD\_YORG<n>****Description**

Define the horizontal and vertical origin of the display (respectively) controlled by the configured driver.

**Type**

Numerical values

**Additionnal information**

In a single display system, both macros are usually set to 0 (the default value).

## 20.5 Color configuration

**LCD\_MAX\_LOG\_COLORS****Description**

Defines the maximum number of colors supported by the driver in a bitmap.

**Type**

Numerical value (default is 256)

**Additionnal information**

If you are using a 4-grayscale LCD, it is usually sufficient to set this value to 4. However, in this case remember not to try to display bitmaps with more than 4 colors.

**LCD\_PHYSCOLORS****Description**

Defines the contents of the color lookup table, if one is used.

**Type**

Alias

**Additionnal information**

This macro is only required if `LCD_FIXEDPALETTE` is set to 0. Refer to the color section for more information.

**LCD\_PHYSCOLORS\_IN\_RAM****Description**

Puts the contents of the physical color table in RAM if enabled.

**Type**

Binary switch  
0: inactive (default)  
1: active

**LCD\_REVERSE****Description**

Inverts the display at compile time.

**Type**

Binary switch  
0: inactive, not reversed (default)  
1: active, reversed

**LCD\_SWAP\_RB****Description**

Swaps the red and blue color components.

**Type**

Binary switch  
0: inactive, not swapped (default)  
1: active, swapped

## 20.6 Magnifying the LCD

Some hardware requires the LCD to be magnified in order to display images correctly. The software must compensate for hardware which internally needs magnification. It does so by activating a layer (above the driver layer) to automatically handle the job of magnifying the display.

**LCD\_XMAG****Description**

Specifies the horizontal magnification factor of the LCD.

**Type**

Numerical value (default is 1)

**Additionnal information**

A factor of 1 results in no magnification.

**LCD\_YMAG****Description**

Specifies the vertical magnification factor of the LCD.

**Type**

Numerical value (default is 1)

**Additionnal information**

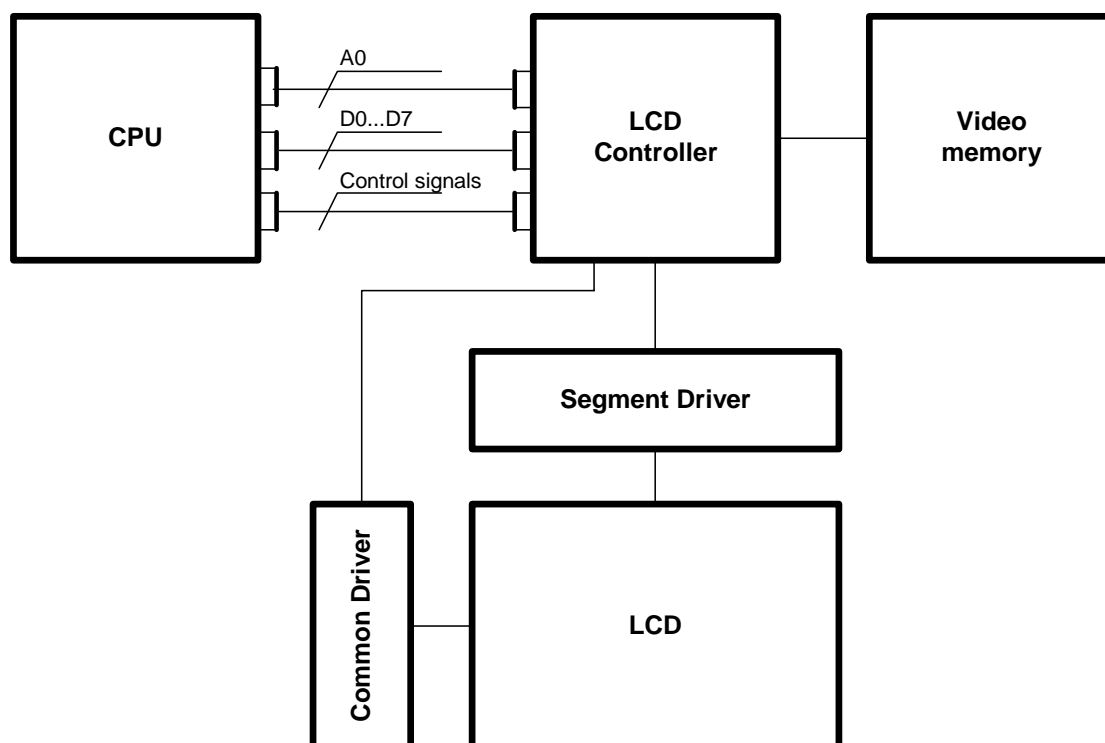
A factor of 1 results in no magnification.

## 20.7 Simple bus interface configuration

There are basically 2 types of bus interface for LCD controllers: full- and simple bus interfaces.

Most LCD controllers for smaller displays (usually up to 240\*128 or 320\*240) use a simple bus interface to connect to the CPU. With a simple bus, only one address bit (usually A0) is connected to the LCD controller. Some of these controllers are very slow, so that the hardware designer may decide to connect it to input/output (I/O) pins instead of the address bus.

### Block diagram for LCD controllers with simple bus interface



Eight data bits, one address bit and 2 or 3 control lines are used to connect the CPU and one LCD controller. Four macros inform the LCD driver how to access each controller used. If the LCD controller(s) is connected directly to the address bus of the CPU, configuration is simple and usually consists of no more than one line per macro. If the LCD controller(s) is connected to I/O pins, the bus interface must be simulated, which takes about 5-10 lines of program per macro (or a function call to a routine which simulates the bus interface).

The following macros are used only for LCD controllers with simple bus interface.

### LCD\_READ\_A0

#### Description

Reads a byte from LCD controller with A0 (C/D) - line low.

**Type**

Function replacement

**Prototype**

```
#define LCD_READ_A0(Result)
```

Parameter	Meaning
<a href="#">Result</a>	Result read. This is not a pointer, but a placeholder for the variable in which the value will be stored.

**LCD\_READ\_A1****Description**

Reads a byte from LCD controller with A0 (C/D) - line high.

**Type**

Function replacement

**Prototype**

```
#define LCD_READ_A1(Result)
```

Parameter	Meaning
<a href="#">Result</a>	Result read. This is not a pointer, but a placeholder for the variable in which the value will be stored.

**LCD\_WRITE\_A0****Description**

Writes a byte to LCD controller with A0 (C/D) - line low.

**Type**

Function replacement

**Prototype**

```
#define LCD_WRITE_A0(Byte)
```

Parameter	Meaning
<a href="#">Byte</a>	Byte to write.

**LCD\_WRITE\_A1****Description**

Writes a byte to LCD controller with A0 (C/D) - line high.

**Type**

Function replacement

**Prototype**

```
#define LCD_WRITE_A1(Byte)
```

Parameter	Meaning
<a href="#">Byte</a>	Byte to write.

## LCD\_WRITEM\_A1

### Description

Writes several bytes to the LCD controller with A0 (C/D) - line high.

### Type

Function replacement

### Prototype

```
#define LCD_WRITEM_A1(paBytes, NumberOfBytes)
```

Parameter	Meaning
<code>paBytes</code>	Placeholder for the pointer to the first data byte.
<code>NumberOfBytes</code>	Number of data bytes to be written.

## Example of real bus interface

The following example demonstrates how to access the LCD by a real bus interface:

```
void WriteM_A1(char *paBytes, int NummerOfBytes) {
    int i;
    for (i = 0; i < NummerOfBytes; i++) {
        (*(volatile char *)0xc0001) = *(paBytes + i);
    }
}

#define LCD_READ_A1(Result) Result = (*(volatile char *)0xc0000)
#define LCD_READ_A0(Result) Result = (*(volatile char *)0xc0001)
#define LCD_WRITE_A1(Byte) (*(volatile char *)0xc0000) = Byte
#define LCD_WRITE_A0(Byte) (*(volatile char *)0xc0001) = Byte

#define LCD_WRITEM_A1(paBytes, NummerOfBytes) WriteM_A1(paBytes, NummerOfBytes)
```

## Sample routines for connection to I/O pins

Several examples can be found in the folder Sample\LCD\_X:

- Port routines for 6800 interface
- Port routines for 8080 interface
- Simple port routines for a serial interface
- Port routines for a simple I2C bus interface

These samples can be used directly. All you need to do is to define the port access macros listed at the top of each example and to map them in your LCDConf.h in a similar manner to that shown below:

```
void LCD_X_Write00(char c);
void LCD_X_Write01(char c);
char LCD_X_Read00(void);
char LCD_X_Read01(void);
#define LCD_WRITE_A1(Byte) LCD_X_Write01(Byte)
#define LCD_WRITE_A0(Byte) LCD_X_Write00(Byte)
#define LCD_READ_A1(Result) Result = LCD_X_Read01()
#define LCD_READ_A0(Result) Result = LCD_X_Read00()
```

Note that not all LCD controllers handle the A0 or C/D bit in the same way. For example, a Toshiba controller requires that this bit be low when accessing data and an Epson SED1565 requires it to be high.

## Hardware access for multiple LCD controllers

If more than one LCD controller is used for the LCD, you must define access macros for each of them individually, according to your hardware. Four macros are needed per LCD controller. The macros for the additional controllers are often very similar to those for the first one. With a direct bus connection, usually only the addresses are different. When I/O pins are used, the sequence for the access is the same except for the CHIP-SELECT signal.

When using more than one controller, add an underscore and the index of the controller as the postfix. For example:

Controller #0: `LCD_READ_A0_0`

Controller #1: `LCD_READ_A0_1`

and so on.

Note that the first controller is considered to be controller #0, so that a second controller would be defined as #1, etc. The macros for additional LCD controllers are listed as follows.

### Second LCD controller

Type	Macro	Explanation
F	<code>LCD_READ_A0_1(Result)</code>	LCD controller 1: Read a byte with A0 = 0.
F	<code>LCD_READ_A1_1(Result)</code>	LCD controller 1: Read a byte with A0 = 1.
F	<code>LCD_WRITE_A0_1(Byte)</code>	LCD controller 1: Write a byte with A0 = 0.
F	<code>LCD_WRITE_A1_1(Byte)</code>	LCD controller 1: Write a byte with A0 = 1.

### Third LCD controller

Type	Macro	Explanation
F	<code>LCD_READ_A0_2(Result)</code>	LCD controller 2: Read a byte with A0 = 0.
F	<code>LCD_READ_A1_2(Result)</code>	LCD controller 2: Read a byte with A0 = 1.
F	<code>LCD_WRITE_A0_2(Byte)</code>	LCD controller 2: Write a byte with A0 = 0.
F	<code>LCD_WRITE_A1_2(Byte)</code>	LCD controller 2: Write a byte with A0 = 1.

### Fourth LCD controller

Type	Macro	Explanation
F	<code>LCD_READ_A0_3(Result)</code>	LCD controller 3: Read a byte with A0 = 0.
F	<code>LCD_READ_A1_3(Result)</code>	LCD controller 3: Read a byte with A0 = 1.
F	<code>LCD_WRITE_A0_3(Byte)</code>	LCD controller 3: Write a byte with A0 = 0.
F	<code>LCD_WRITE_A1_3(Byte)</code>	LCD controller 3: Write a byte with A0 = 1.

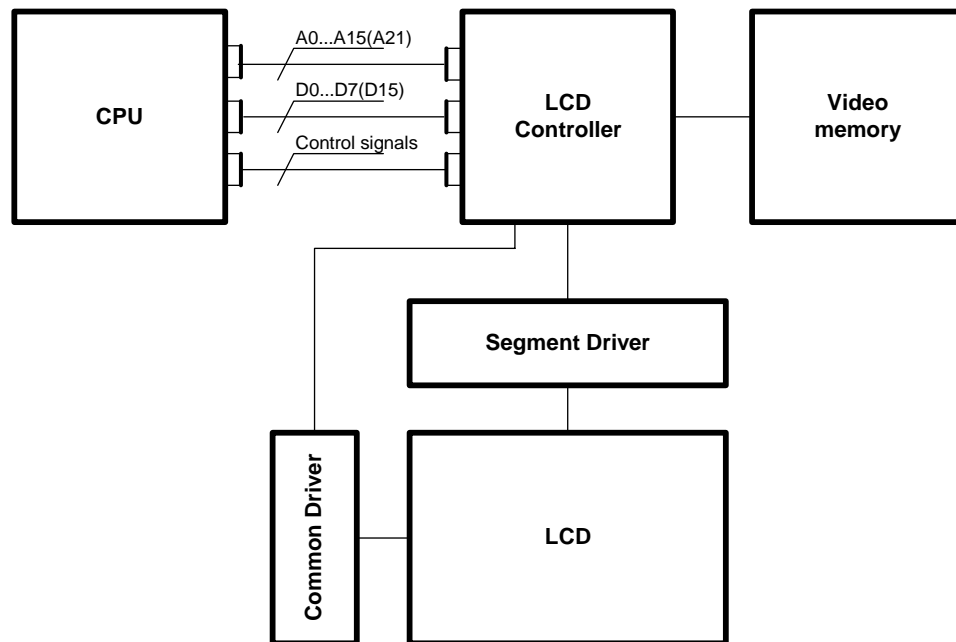
## 20.8 Full bus interface configuration

Some LCD controllers (especially those for displays with higher resolution) require a full-address bus, which means they are connected to at least 14 address bits. In a full bus interface configuration, video memory is directly accessible by the CPU; the full-address bus is connected to the LCD controller.

The only knowledge required when configuring a full bus interface is information about the address range (which will generate a CHIP-SELECT signal for the LCD controller) and whether 8- or 16-bit accesses should be used (bus-width to the LCD controller). In other words, you need to know the following:

- Base address for video memory access
- Base address for register access
- Distance between adjacent video memory locations (usually 1/2/4-byte)
- Distance between adjacent register locations (usually 1/2/4-byte)
- Type of access (8/16/32-bit) for video memory
- Type of access (8/16/32-bit) for registers

### Typical block diagram for LCD controllers with full bus interface



### Configuration example

The example assumes the following:

Base address video memory	0x80000
Base address registers	0xc0000
Access to video RAM	16-bit
Access to register	16-bit
Distance between adjacent video memory locations	2 bytes
Distance between adjacent register locations	2 bytes

```

#define LCD_READ_REG(Index)      *((U16*)(0xc0000+(Off<<1)))
#define LCD_WRITE_REG(Index,data) *((U16*)(0xc0000+(Off<<1)))=data
#define LCD_READ_MEM(Index)     *((U16*)(0x80000+(Off<<1)))
#define LCD_WRITE_MEM(Index,data) *((U16*)(0x80000+(Off<<1)))=data
  
```

The following macros are used only for LCD controllers with full bus interface.

## LCD\_READ\_MEM

### Description

Reads the video memory of the LCD controller.

### Type

Function replacement

### Prototype

```
#define LCD_READ_MEM(Index)
```

Parameter	Meaning
<a href="#">Index</a>	Index of video memory of controller.

### Additional information

This macro defines how to read the video memory of the LCD controller.

In order to configure this switch correctly, you need to know the base address of the video memory, the spacing and if 8/16- or 32-bit accesses are permitted. You should also know the correct syntax for your compiler because this kind of hardware access is not defined in ANSI "C" and is therefore different for different compilers.

## LCD\_READ\_REG

### Description

Reads the register of the LCD controller.

### Type

Function replacement

### Prototype

```
#define LCD_READ_REG(Index)
```

Parameter	Meaning
<a href="#">Index</a>	Index of the register to read.

### Additional information

This macro defines how to read the registers of the LCD controller. Usually, the registers are memory-mapped. In this case, the macro can normally be written as a single line.

In order to configure this switch correctly, you need to know the address the registers are mapped to, the spacing and if 8/16- or 32-bit accesses are permitted. You should also know the correct syntax for your compiler because this kind of hardware access is not defined in ANSI "C" and is therefore different for different compilers. However, the syntax shown below works with the majority of them.

### Example

If the registers are mapped to a memory area starting at 0xc0000, the spacing is 2 and 16-bit accesses should be used; with most compilers the define should look as follows:

```
#define LCD_READ_REG(Index) *((U16*)(0xc0000+(Off<<1)))
```

## LCD\_WRITE\_MEM

### Description

Writes data to the video memory of the LCD controller.

### Type

Function replacement

### Prototype

LCD\_WRITE\_MEM(Index, Data)

Parameter	Meaning
<a href="#">Index</a>	Index of video memory of controller.
<a href="#">Data</a>	Data to write to the register

### Additional information

This macro defines how to write to the video memory of the LCD controller.

In order to configure this switch correctly, you need to know the base address of the rvideo memory, the spacing and if 8/16- or 32-bit accesses are permitted, as well as the correct syntax for your compiler.

With 8-bit accesses, a value of 1 indicates byte 1.

With 16-bit accesses, a value of 1 indicates word 1.

## LCD\_WRITE\_REG

### Description

Writes data to a specified register of the LCD controller

### Type

Function replacement

### Prototype

LCD\_WRITE\_REG(Index, Data)

Parameter	Meaning
<a href="#">Index</a>	Index of the register to write to
<a href="#">Data</a>	Data to write to the register

### Additional information

This macro defines how to write to the registers of the LCD controller. If the registers are memory-mapped, the macro can normally be written as a single line.

In order to configure this switch correctly, you need to know the address the registers are mapped to, the spacing and if 8/16- or 32-bit accesses are permitted, as well as the correct syntax for your compiler.

With 8-bit accesses, a value of 1 indicates byte 1.

With 16-bit accesses, a value of 1 indicates word 1.

### Example

If the registers are mapped to a memory area starting at 0xc0000, the spacing is 4 and 8-bit access should be used; with most compilers the define should look as follows:

```
#define LCD_WRITE_REG(Index,Data) *((U8volatile *) (0xc0000+(Off<<2)))=data
```

## LCD\_BUSWIDTH

### Description

Defines bus-width of LCD controller/CPU interface (external display access).

### Type

Selection switch

8: 8-bit wide VRAM

16: 16-bit wide VRAM (default)

### Additional information

Since this completely depends on your hardware, you will have to substitute these macros. The Epson SED1352 distinguishes between memory and register access; memory is the video memory of the LCD controller and registers are the 15 configuration registers. The macros define how to access (read/write) VRAM and registers.

## LCD\_ENABLE\_REG\_ACCESS

### Description

Enables register access and sets the M/R signal to high.

### Type

Function replacement

### Prototype

```
#define LCD_ENABLE_REG_ACCESS() MR = 1
```

### Additional information

Only used for Epson SED1356 and SED1386 controllers.

After using this macro, `LCD_ENABLE_MEM_ACCESS` must also to be defined in order to switch back to memory access after accessing the registers.

## LCD\_ENABLE\_MEM\_ACCESS

### Description

Switches the M/R signal to memory access. It is executed after register access functions and sets the M/R signal to low.

### Type

Function replacement

### Prototype

```
#define LCD_ENABLE_MEM_ACCESS() MR = 0
```

### Additional information

Only used for Epson SED1356 and SED1386 controllers.

## LCD\_SWAP\_BYTE\_ORDER

### Description

Inverts the endian mode (swaps the high and low bytes) between CPU and LCD controller when using a 16-bit bus interface.

### Type

Binary switch

0: inactive, endian modes not swapped (default)  
 1: active, endian modes swapped

## 20.9 LCD controller configuration: common/segment lines

For most LCDs, the setup of common (COM) and segment (SEG) lines is straightforward and neither special settings for COM/SEG lines nor the configuration macros in this section are required.

This section explains how the LCD controller(s) is physically connected to your display. The direction does not matter; it is only assumed that continuous COM and SEG lines are used. If the direction of SEGs or COMs is reversed, use `LCD_MIRROR_X/LCD_MIRROR_Y` to set them in the direction required by your application. If non-continuous COM/SEG lines have been used, you have to modify the driver (putting in a translation table will do) or -- even better -- go back to the hardware (LCD module) designer and ask him/her to start over.

### **LCD\_XORG<n>; LCD\_YORG<n>**

#### **Description**

Define the horizontal and vertical origin of the display (respectively) controlled by the configured driver.

#### **Type**

Numerical value

#### **Additional information**

In a single display system, both macros are usually set to 0 (the default value).

### **LCD\_FIRSTSEG<n>**

#### **Description**

Controller <n>: first segment line used.

#### **Type**

Numerical value

### **LCD\_LASTSEG<n>**

#### **Description**

Controller <n>: last segment line used.

#### **Type**

Numerical value

### **LCD\_FIRSTCOM<n>**

#### **Description**

Controller <n>: first common line used.

#### **Type**

Numerical value

## LCD\_LASTCOM<n>

### Description

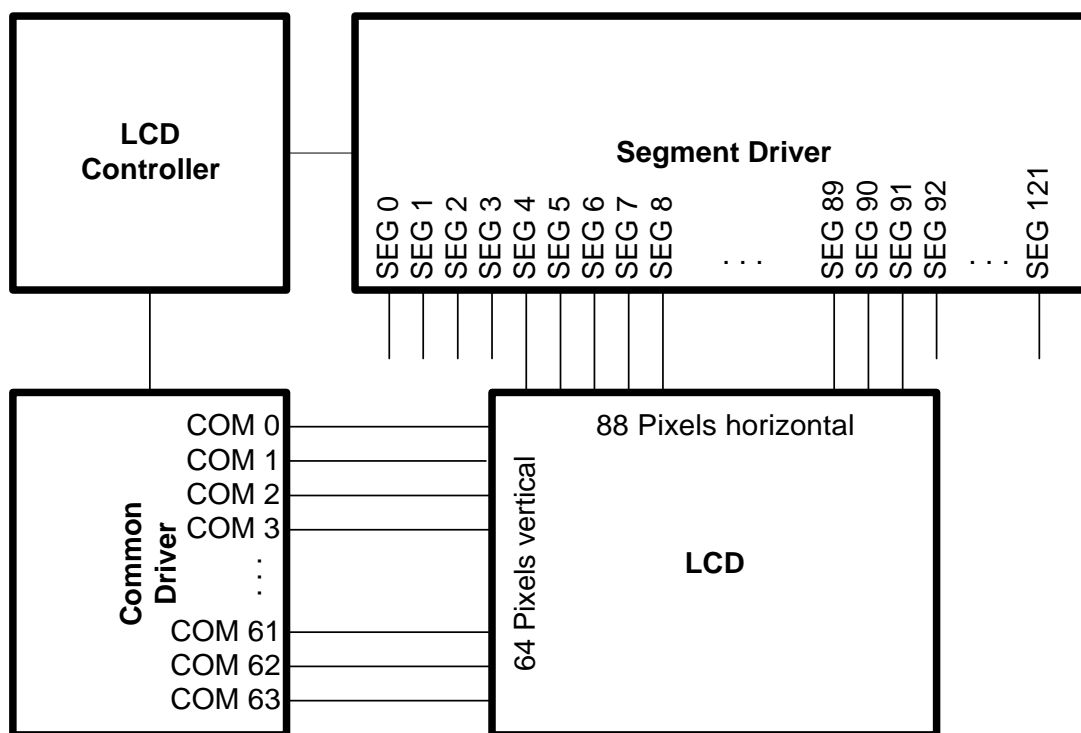
Controller <n>: last common line used.

### Type

Numerical value

## Single LCD controller configuration

The following block diagram shows a single LCD, controlled by a single LCD controller, using external COM and SEG drivers. All outputs of the common driver (COM0-COM63) are being used, but only some outputs of the segment driver (SEG4-SEG91). Note that for simplicity the video RAM is not shown in the diagram.



### Configuration for the above example

```

#define LCD_FIRSTSEG0    (4)      /* Contr.0: first segment line used */
#define LCD_LASTSEG0     (91)     /* Contr.0: last segment line used */
#define LCD_FIRSTCOM0    (0)      /* Contr.0: first com line used */
#define LCD_LASTCOM0     (63)     /* Contr.0: last com line used */
  
```

Please also note that the above configuration is identical if the COM or SEG lines are mirrored and even if the LCD is built-in sideways (90° turned, X-Y swapped). The same applies if the COM/SEG drivers are integrated into the LCD controller, as is the case for some controllers designed for small LCDs. A typical example for this type of controller would be the Epson SED15XX series.

## Configuring additional LCD controllers

μC/GUI offers the possibility of using more than one (up to four) LCD controllers with one LCD. The configuration switches are identical to the switches for the first controller (controller 0), except the index is 1, 2 or 3 instead of 0.

### Second LCD controller

Type	Macro	Explanation
N	LCD_FIRSTSEG1	LCD controller 1: first segment line used.
N	LCD_LASTSEG1	LCD controller 1: last segment line used.
N	LCD_FIRSTCOM1	LCD controller 1: first com line used.
N	LCD_LASTCOM1	LCD controller 1: last com line used.
N	LCD_XORG1	LCD controller 1: leftmost (lowest) X-position.
N	LCD_YORG1	LCD controller 1: topmost (lowest) Y-position.

### Third LCD controller

Type	Macro	Explanation
N	LCD_FIRSTSEG2	LCD controller 2: first segment line used.
N	LCD_LASTSEG2	LCD controller 2: last segment line used.
N	LCD_FIRSTCOM2	LCD controller 2: first com line used.
N	LCD_LASTCOM2	LCD controller 2: last com line used.
N	LCD_XORG2	LCD controller 2: leftmost (lowest) X-position.
N	LCD_YORG2	LCD controller 2: topmost (lowest) Y-position.

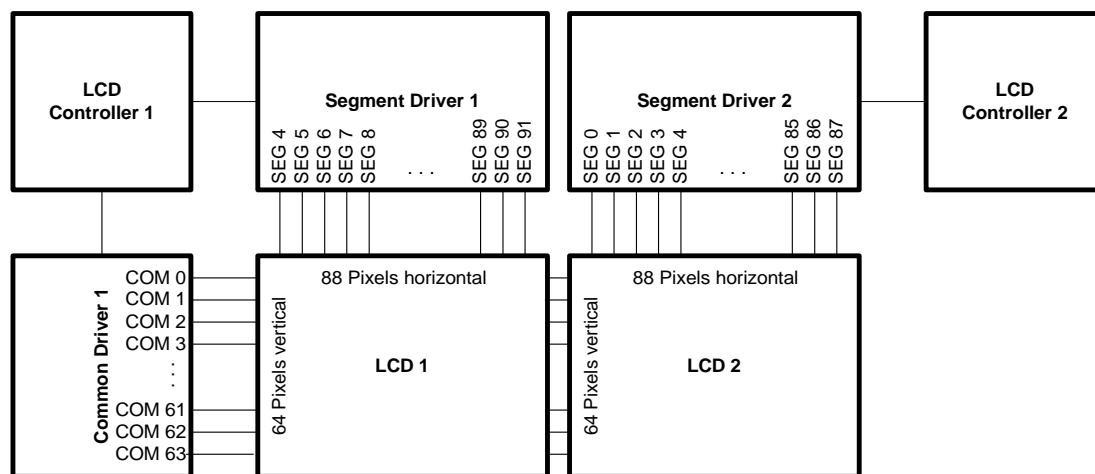
### Fourth LCD controller

Type	Macro	Explanation
N	LCD_FIRSTSEG3	LCD controller 3: first segment line used.
N	LCD_LASTSEG3	LCD controller 3: last segment line used.
N	LCD_FIRSTCOM3	LCD controller 3: first com line used.
N	LCD_LASTCOM3	LCD controller 3: last com line used.
N	LCD_XORG3	LCD controller 3: leftmost (lowest) X-position.
N	LCD_YORG3	LCD controller 3: topmost (lowest) Y-position.

When using more than one LCD controller, be sure to remember to define the number of controllers used (see macro `LCD_NUM_CONTROLLERS` in the next section).

### Example

The following diagram shows a hardware configuration using two LCD controllers. The COM lines are driven by the common driver connected to controller 1 and are directly connected to the second LCD. LCD 1 is connected to segment driver 1 using SEG lines 4 to 91. LCD 2 is driven by SEG 0 to SEG 87 of segment driver 2.



### Configuration for the above example

```
#define LCD_FIRSTSEG0      (4)      /* Contr.0: first segment line used */
#define LCD_LASTSEG0      (91)     /* Contr.0: last segment line used */
#define LCD_FIRSTCOM0     (0)      /* Contr.0: first com line used */
#define LCD_LASTCOM0      (63)     /* Contr.0: last com line used */
#define LCD_XORG0         (0)      /* Contr.0: leftmost (lowest) x-Pos */
#define LCD_YORG0         (0)      /* Contr.0: topmost (lowest) y-Pos */
#define LCD_FIRSTSEG1     (0)      /* Contr.1: first segment line used */
#define LCD_LASTSEG1      (87)     /* Contr.1: last segment line used */
#define LCD_FIRSTCOM1     (0)      /* Contr.1: first com line used */
#define LCD_LASTCOM1      (63)     /* Contr.1: last com line used */
#define LCD_XORG1         (88)     /* Contr.1: leftmost (lowest) x-Pos */
#define LCD_YORG1         (0)      /* Contr.1: topmost (lowest) y-Pos */
```

## 20.10 COM/SEG lookup tables

When using "chip on glass" technology, it is sometimes very difficult to ensure that the COM and SEG outputs of the controller(s) are connected to the display in a linear fashion. In this case a COM/SEG lookup table may be required in order to inform the driver as to how the COM/SEG lines are connected.

### LCD\_LUT\_COM

#### Description

Defines a COM lookup table for the controller.

#### Type

Alias

#### Example

Let us assume your display contains only 10 COM lines and their connecting order is 0, 1, 2, 6, 5, 4, 3, 7, 8, 9. To configure the LCD driver so that the COM lines are accessed in the correct order, the following macro should be added to your LCD-Conf.h:

```
#define LCD_LUT_COM 0, 1, 2, 6, 5, 4, 3, 7, 8, 9
```

If you need to modify the segment order, you should use the macro LCD\_LUT\_SEG in the same manner.

## LCD\_LUT\_SEG

### Description

Defines a SEG lookup table for the controller.

### Type

Alias

## 20.11 Miscellaneous

### LCD\_NUM\_CONTROLLERS

#### Description

Defines the number of LCD controllers used.

#### Type

Numerical value (default is 1)

### LCD\_CACHE

#### Description

Controls caching of video memory in CPU memory.

#### Type

Binary switch

0: disabled, no display data cache used

1: enabled, display data cache used (default)

#### Additional information

This switch is not supported by all LCD drivers.

Using a display data cache (which speeds up access) is recommended if access to the video memory is slow, which is usually the case with larger displays and simple bus interfaces (particularly if port-access or serial interfaces are used). Disabling the cache will slow down the speed of the driver.

### LCD\_USE\_BITBLT

#### Description

Controls usage of hardware acceleration.

#### Type

Binary switch

0: disabled, BitBLT engine is not used

1: enabled, BitBLT engine is used (default)

#### Additional information

Disabling the BitBLT engine will instruct the driver not to use the available hardware acceleration.

### LCD\_SUPPORT\_CACHECONTROL

#### Description

Switch support for the `LCD_L0_ControlCache()` function of the driver.

**Type**

Binary switch

0: disabled, `LCD_L0_ControlCache()` may not be used (default)

1: enabled, `LCD_L0_ControlCache()` may be used

**Additional information**

The API function `LCD_L0_ControlCache()` permits locking, unlocking, or flushing of the cache. For more information, refer to Chapter 23: "LCD Driver API".

Please note that this feature is intended only for some LCD controllers with simple bus interface, for which it is important to access the controller as little as possible in order to maximize speed. For other controllers, this switch has no effect.

**LCD\_TIMERINIT0****Description**

Timing value used by an interrupt service routine for displaying pane 0 of a pixel.

**Type**

Numerical value

**Additional information**

This macro is only relevant when no LCD controller is used, since it is then the job of the CPU to update the display in an interrupt service routine.

**LCD\_TIMERINIT1****Description**

Timing value used by an interrupt service routine for displaying pane 1 of a pixel.

**Type**

Numerical value

**Additional information**

This macro is only relevant when no LCD controller is used, since it is then the job of the CPU to update the display in an interrupt service routine.

**LCD\_ON****Description**

Switches the LCD on.

**Type**

Function replacement

**LCD\_OFF****Description**

Switches the LCD off.

**Type**

Function replacement

µC/GUI

# Chapter 21

## High-Level Configuration

---

High-level configuration is relatively simple. In the beginning, you can normally use the existing configuration files (for example, those used in the simulation). Only if there is a need to fine-tune the system, or to minimize memory consumption, does the high-level configuration file `GUIConf.h` need to be changed. This file is usually located in the `Config` subdirectory of your project's root directory. Use the file `GUIConf.h` for any high-level configuration.

The second thing to do when using  $\mu$ C/GUI on your hardware is to change the hardware-dependent functions, located in the file `Sample\GUI_X\GUI_X.c`.

## 21.1 Available GUI configuration macros

The following table shows the available macros used for high-level configuration of  $\mu$ C/GUI

Type	Macro	Default	Explanation
N	GUI_ALLOC_SIZE	1000	Define the size (number of bytes available) for optional dynamic memory. Dynamic memory is required only for windows and memory devices. This memory will only be used if the GUIAlloc module is linked, which should happen only if dynamic memory is required.
S	GUI_DEBUG_LEVEL	---	Define which information has to be transmitted by GUI_X_Log(). You will find a detailed description later in the chapter under GUI_X_Log().
N	GUI_DEFAULT_BKCOLOR	GUI_BLACK	Define the default background color.
N	GUI_DEFAULT_COLOR	GUI_WHITE	Define the default foreground color.
S	GUI_DEFAULT_FONT	&GUI_Font6x8	Define which font is used as default after GUI_Init(). If you do not use the default font, it makes sense to change to a different default, as the default font is referenced by the code and will therefore always be linked.
N	GUI_MAXTASK	4	Define the maximum number of tasks from which $\mu$ C/GUI is called to access the display when multitasking support is enabled (see Chapter 11: "Execution Model: Single Task/Multitask").
B	GUI_OS	0	Activate to enable multitasking support with multiple tasks calling $\mu$ C/GUI (see Chapter 11: "Execution Model: Single Task/Multitask").
B	GUI_SUPPORT_MEMDEV	1	Enables optional memory device support. Not using memory devices will save about 40 bytes of RAM for function pointers and will slightly accelerate execution.
B	GUI_SUPPORT_TOUCH	0	Enables optional touch-screen support.
B	GUI_SUPPORT_UNICODE	1	Enables support for Unicode characters embedded in 8-bit strings. Please note: Unicode characters may always be displayed, as character codes are always treated as 16-bit.
B	GUI_WINSUPPORT	0	Enables optional window manager support.

### How to configure the GUI

We recommend the following procedure:

1. Make a copy of the original configuration file.
2. Review all configuration switches.
3. Erase unused sections of the configuration.

### Sample configuration

The following is a short sample GUI configuration file (ConfigSample\GUIConf.h):

```

/*****
*
*      Configuration of desired functionality
*
*****/

#define GUI_WINSUPPORT          (1)  /* Use window manager if true (1)  */
#define GUI_SUPPORT_TOUCH      (1)  /* Support a touch screen */

/*****
*
*      Configuration of dynamic memory
*
*****/

Dynamic memory is used for memory devices and window manager.
If you do not use these features, there is no need for dynamic memory
and it may be switched off completely. (This section can be erased)
*/

#ifndef GUI_ALLOC_SIZE
#define GUI_ALLOC_SIZE          5000 /* Size of dynamic memory */
#endif

/*****
*
*      Configuration of fonts
*
*****/

If your create additionnal fonts (usually using the  $\mu$ C/GUI-FontConvert), these
fonts need to be declared as extern in order to be able to use them.
This would be a good place to do this.
*/

#define GUI_DEFAULT_FONT      &GUI_Font6x8 /* This font is used as default */

```

## 21.2 GUI\_X routine reference

When using  $\mu$ C/GUI with your hardware, there are several hardware-dependent functions which must exist in your project. When using the simulation, the library already contains them. A sample file can be found under `Sample\GUI_X\GUI_X.c`. The following table lists the available hardware-dependent functions in alphabetical order within their respective categories. Detailed description of the routines can be found in the sections that follow.

Routine	Explanation
Init routine	
<a href="#">GUI_X_Init()</a>	Called from <code>GUI_Init()</code> ; can be used to initialize hardware.
Timing routines	
<a href="#">GUI_X_Delay()</a>	Return after a given period.
<a href="#">GUI_X_ExecIdle()</a>	Called only from non-blocking functions of window manager.
<a href="#">GUI_X_GetTime()</a>	Return the system time in milliseconds.
Kernel interface routines	
<a href="#">GUI_X_InitOS()</a>	Initialize the kernel interface module (create a resource semaphore/mutex).
<a href="#">GUI_X_GetTaskId()</a>	Return a unique, 32-bit identifier for the current task/thread.
<a href="#">GUI_X_Lock()</a>	Lock the GUI (block resource semaphore/mutex).

Routine	Explanation
<code>GUI_X_Unlock()</code>	Unlock the GUI (unblock resource semaphore/mutex).
Debugging	
<code>GUI_X_Log()</code>	Return debug information; required if logging is enabled.

## 21.3 Init routine

### GUI\_X\_Init()

#### Description

Called from `GUI_Init()`; can be used to initialize hardware.

#### Prototype

```
void GUI_X_Init(void);
```

## 21.4 Timing routines

### GUI\_X\_Delay()

#### Description

Returns after a specified time period in milliseconds.

#### Prototype

```
void GUI_X_Delay(int Period)
```

Parameter	Meaning
<code>Period</code>	Period in milliseconds.

### GUI\_X\_ExecIdle()

#### Description

Called only from non-blocking functions of the window manager.

#### Prototype

```
void GUI_X_ExecIdle(void);
```

#### Additional information

Called when there are no longer any messages which require processing. In this case the GUI is up to date.

### GUI\_X\_GetTime()

#### Description

Used by `GUI_GetTime` to return the current system time in milliseconds.

#### Prototype

```
int GUI_X_GetTime(void)
```

### Return value

The current system time in milliseconds, of type integer.

## 21.5 Kernel interface routines

Detailed descriptions for these routines may be found in Chapter 11: "Execution Model: Single Task/Multitask".

## 21.6 Debugging

### GUI\_X\_Log()

#### Description

Returns debug information from  $\mu$ C/GUI.

#### Prototype

```
void GUI_X_Log(const char * s);
```

Parameter	Meaning
<a href="#">s</a>	Pointer to the string to be sent.

#### Additional information

This routine is called by  $\mu$ C/GUI to transmit error messages or warnings, and is required if logging is enabled. The GUI calls this function depending on the configuration macro `GUI_DEBUG_LEVEL`. The following table lists the permitted values for `GUI_DEBUG_LEVEL`:

Value	Explanation
<a href="#">GUI_DEBUG_LEVEL_NOCHECK</a>	No run-time checks are performed.
<a href="#">GUI_DEBUG_LEVEL_CHECK_PARA</a>	Parameter checks are performed to avoid crashes.
<a href="#">GUI_DEBUG_LEVEL_CHECK_ALL</a>	Parameter checks and consistency checks are performed.
<a href="#">GUI_DEBUG_LEVEL_LOG_ERRORS</a>	Errors are recorded.
<a href="#">GUI_DEBUG_LEVEL_LOG_WARNINGS</a>	Errors and warnings are recorded.
<a href="#">GUI_DEBUG_LEVEL_LOG_ALL</a>	Errors, warnings and messages are recorded.

$\mu$ C/GUI



# Chapter 22

## LCD Drivers

---

An LCD driver supports a particular family of LCD controllers and all LCDs which are equipped with one or more of these controllers. The driver is essentially generic, meaning it can be configured by modifying the configuration file `LCDConf.h`. This file contains all configurable options for the driver as well as multiple defines for how the hardware is accessed and how the controller(s) is connected to the LCD.

This chapter provides an overview of the LCD drivers available for  $\mu$ C/GUI. It explains the following in terms of each driver:

- Which LCD controllers can be accessed, as well as supported color depths and types of interfaces.
- Additional RAM requirements.
- Additional functions.
- How to access the hardware.
- Special configuration switches.
- Special requirements for particular LCD controllers.

## 22.1 Supported LCD controllers and respective drivers

The following table lists the available drivers and which LCD controllers are supported by each:

Driver	Value for macro LCD_CONTROLLER	LCD Controller	Supported bits/pixel
LCD07X1	711 741	Samsung KS0711 Samsung KS0741	2
LCD13XX	1352 1354 1356 1374 1375 1376 1386 1300	Epson SED1352, S1D13502 Epson SED1354, S1D13504 Epson SED1356, S1D13506 Epson SED1374, S1D13704 Epson SED1375, S1D13705 Epson SED1376, S1D13706 Epson SED1386, S1D13806 Epson S1D13A03, S1D13A04	1, 2, 4, 8, 16
LCD159A	0x159A	Epson SED159A	8
LCD15E05	0x15E05	Epson S1D15E05	2
LCD15XX	713 1560 1565 1566 1567 1568 1569 1575	Samsung KS0713 Epson SED1560 Epson SED1565 Epson SED1566 Epson SED1567 Epson SED1568 Epson SED1569 Epson SED1575	1
LCD6642X	66420 66421	Hitachi HD66420 Hitachi HD66421	2
LCDMem	0	No controller, writes into RAM (monochrome displays)	2
LCDMemC	0	No controller, writes into RAM (color displays)	3, 6
LCDPage1bpp	8811	Philips PCF8810, PCF8811	1
LCDSLin	1330 1335 6963	Epson SED1330 Epson SED1335 Toshiba T6963	1

### Selecting a driver

As described in Chapter 20: "Low-Level Configuration", the macro `LCD_CONTROLLER` defines the LCD controller used. A controller is specified by its appropriate value, listed in the table above.

The following sections discuss each of the available drivers individually.

## 22.2 LCD07X1

### Supported hardware

#### Controllers

This driver has been tested with the following LCD controllers:

- Samsung KS0711
- Samsung KS0741

It should be assumed that it will also work with any controller of similar organization.

#### Bits per pixel

Supported color depth is 2 bpp.

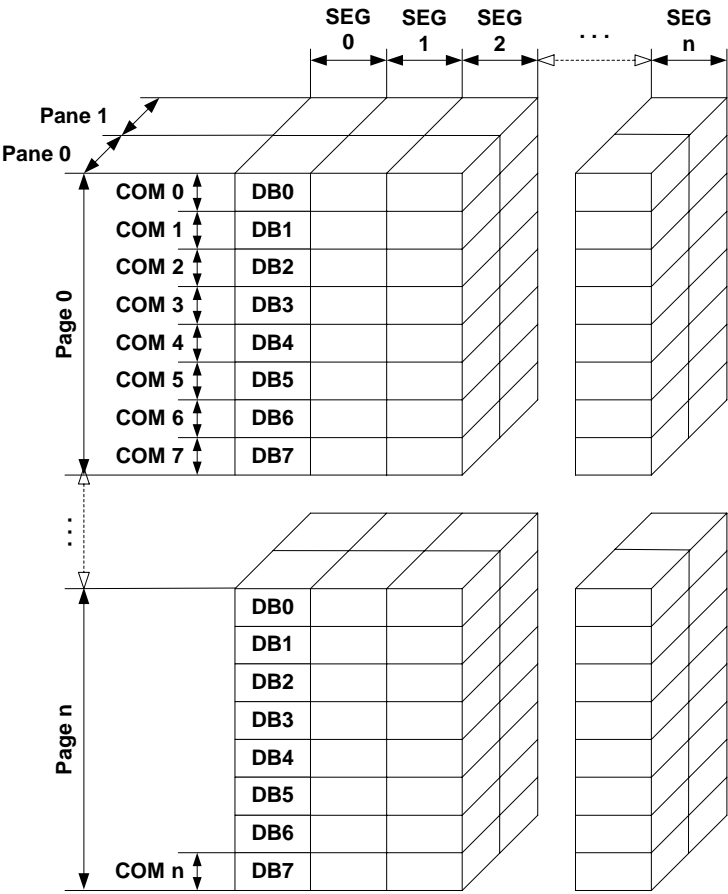
#### Interfaces

The chip supports three types of interfaces:

- 8-bit parallel (simple bus) interface
- 4-pin serial peripheral interface, or SPI.
- 3-pin SPI.

The current version of the driver supports the parallel or 4-pin SPI modes.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD. The display memory is divided into two panes for each pixel. The lower bit of each pixel is stored in pane 0 and the higher bit is stored in pane 1.

Additional RAM requirements of the driver

This LCD driver may be used with or without a display data cache, containing a complete copy of the contents of the LCD data RAM. If a cache is not used, there are no additional RAM requirements.

It is recommended to use this driver with a data cache for faster LCD-access. The amount of memory used by the cache may be calculated as follows:

Size of RAM (in bytes) = (LCD\_YSIZE + 7) / 8 \* LCD\_XSIZE \* 2

Additional driver functions

LCD\_L0\_ControlCache

For information about this function, please refer to Chapter 23: "LCD Driver API".

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 20: "Low-Level Configuration". The following tables lists the macros which must be defined for hardware access:

## Parallel mode

Macro	Explanation
<a href="#">LCD_INIT_CONTROLLER</a>	Initialization sequence for the LCD controller.
<a href="#">LCD_READ_A0</a>	Read a byte from LCD controller with A-line low.
<a href="#">LCD_READ_A1</a>	Read a byte from LCD controller with A-line high.
<a href="#">LCD_WRITE_A0</a>	Write a byte to LCD controller with A-line low.
<a href="#">LCD_WRITE_A1</a>	Write a byte to LCD controller with A-line high.

## Serial mode

Macro	Explanation
<a href="#">LCD_INIT_CONTROLLER</a>	Initialization sequence for the LCD controller.
<a href="#">LCD_WRITE_A0</a>	Write a byte to LCD controller with A-line low.
<a href="#">LCD_WRITE_A1</a>	Write a byte to LCD controller with A-line high.
<a href="#">LCD_WRITE_M_A1</a>	Write multiple bytes to LCD controller with A-line high.

## Additional configuration switches

None.

## Special requirements for certain LCD controllers

None.

# 22.3 LCD13XX

## Supported hardware

### Controllers

This driver has been tested with the following LCD controllers:

- Epson SED1352, S1D13502
- Epson SED1354, S1D13504
- Epson SED1356, S1D13506
- Epson SED1374, S1D13704
- Epson SED1375, S1D13705
- Epson SED1376, S1D13706
- Epson SED1386, S1D13806
- Epson S1D13A03, S1D13A04

It should be assumed that it will also work with any controller of similar organization.

### Bits per pixel

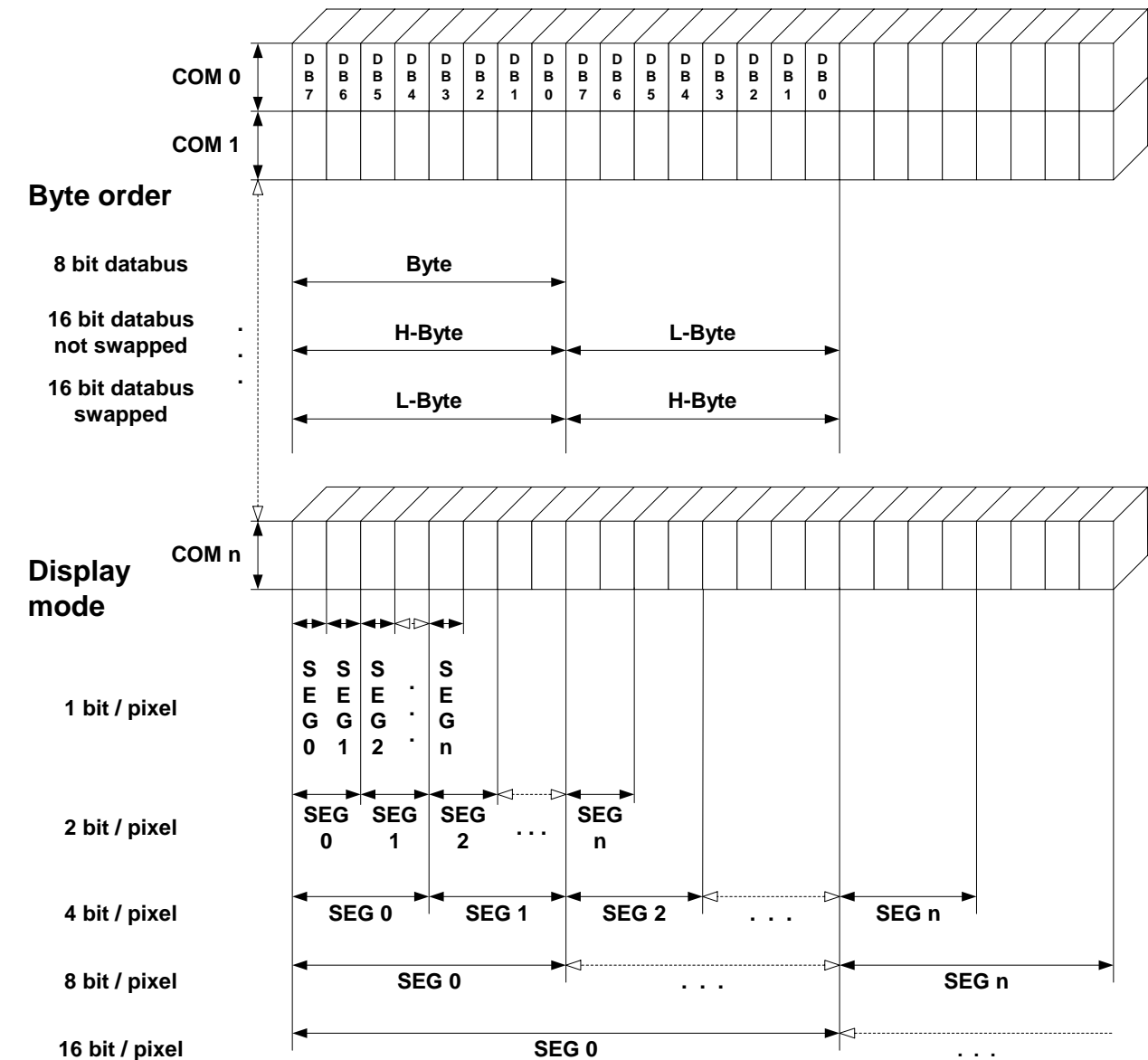
Supported color depths are 1, 2, 4, 8 and 16 bpp.

### Interfaces

The chips supported by this driver can be interfaced in 8/16-bit parallel (full bus) modes.

The driver supports both interfaces. Please refer to the respective LCD controller manual in order to determine if your chip can be interfaced in 8-bit mode.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD in terms of the color depth.

Additional RAM requirements of the driver

None.

Additional driver functions

None.

## Hardware configuration

This driver requires a full bus interface for hardware access as described in Chapter 20: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<a href="#">LCD_INIT_CONTROLLER</a>	Initialization sequence for the LCD controller.
<a href="#">LCD_READ_MEM</a>	Read the contents of video memory of controller.
<a href="#">LCD_READ_REG</a>	Read the contents of a configuration register of controller.
<a href="#">LCD_WRITE_MEM</a>	Write to video memory (display data RAM) of controller.
<a href="#">LCD_WRITE_REG</a>	Write to a configuration register of controller.

## Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
<a href="#">LCD_BUSWIDTH</a>	Select bus-width (8/16) of LCD controller/CPU interface. Default is 16.
<a href="#">LCD_ENABLE_MEM_ACCESS</a>	Switch the M/R signal to memory access. Only used for SED1356 and SED1386 LCD controllers.
<a href="#">LCD_ENABLE_REG_ACCESS</a>	Switch the M/R signal to register access. Only used for SED1356 and SED1386 LCD controllers.
<a href="#">LCD_SWAP_BYTE_ORDER</a>	Inverts the endian mode (swaps the high and low bytes) between CPU and LCD controller when using a 16-bit bus interface.
<a href="#">LCD_USE_BITBLT</a>	If set to 0, it disables the BitBLT engine. If set to 1 (the default value), the driver will use all available hardware acceleration.
<a href="#">LCD_ON</a>	Function replacement macro which switches the LCD on
<a href="#">LCD_OFF</a>	Function replacement macro which switches the LCD off

## Special requirements for certain LCD controllers

### SED1386 or S1D13806

#### LCD\_SWAP\_RB

The configuration switch `LCD_SWAP_RB` (swaps the red and blue components) must be activated (set to 1) by inserting the following line into `LCDCnf.h`:

```
#define LCD_SWAP_RB (1) /* Has to be set */
```

#### LCD\_INIT\_CONTROLLER

When writing or modifying the initialization macro, consider the following:

- To initialize the embedded SDRAM, bit 7 of register 20 (SDRAM initialization bit) must be set to 1 (a minimum of 200  $\mu$ s after reset).
- When the SDRAM initialization bit is set, the actual initialization sequence occurs at the first SDRAM refresh cycle. The initialization sequence requires approximately 16 MCLKs to complete, and memory accesses cannot be made while the initialization is in progress.

For more information, please see the LCD controller documentation.

LCD\_READ\_REG, LCD\_WRITE\_REG

In order for the BitBLT engine to work, the data type of the offset must be unsigned long. This is set with the configuration macros LCD\_READ\_REG and LCD\_WRITE\_REG as follows:

```
#define LCD_READ_REG(Off)      *((volatile U16*)(0x800000+(((U32)(Off))<<1)))
#define LCD_WRITE_REG(Off,Data) *((volatile U16*)(0x800000+(((U32)(Off))<<1)))=Data
```

22.4 LCD159A

Supported hardware

Controllers

This driver has been tested with the following LCD controllers:

- Epson SED159A

It should be assumed that it will also work with any controller of similar organization.

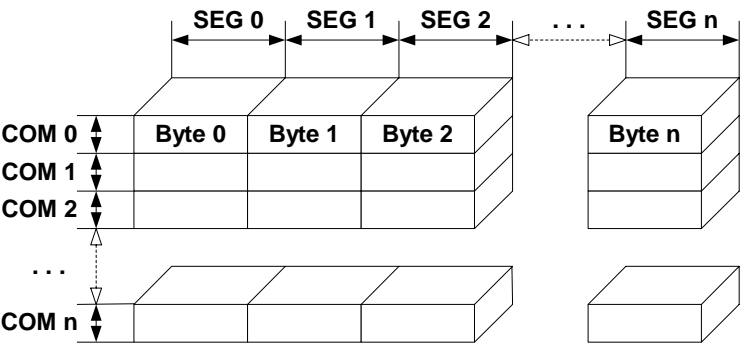
Bits per pixel

Supported color depth is 8 bpp.

Interfaces

The driver supports 8-bit parallel (simple bus) interfaces.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements

None.

Additional driver functions

None.

## Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 20: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<a href="#">LCD_INIT_CONTROLLER</a>	Initialization sequence for the LCD controller.
<a href="#">LCD_READ_A0</a>	Read a byte from LCD controller with A-line low.
<a href="#">LCD_READ_A1</a>	Read a byte from LCD controller with A-line high.
<a href="#">LCD_WRITE_A0</a>	Write a byte to LCD controller with A-line low.
<a href="#">LCD_WRITE_A1</a>	Write a byte to LCD controller with A-line high.

## Additional configuration switches

None.

## Special requirements for certain LCD controllers

None.

# 22.5 LCD15E05

## Supported hardware

### Controllers

This driver has been tested with the following LCD controllers:

- Epson S1D15E05

It should be assumed that it will also work with any controller of similar organization.

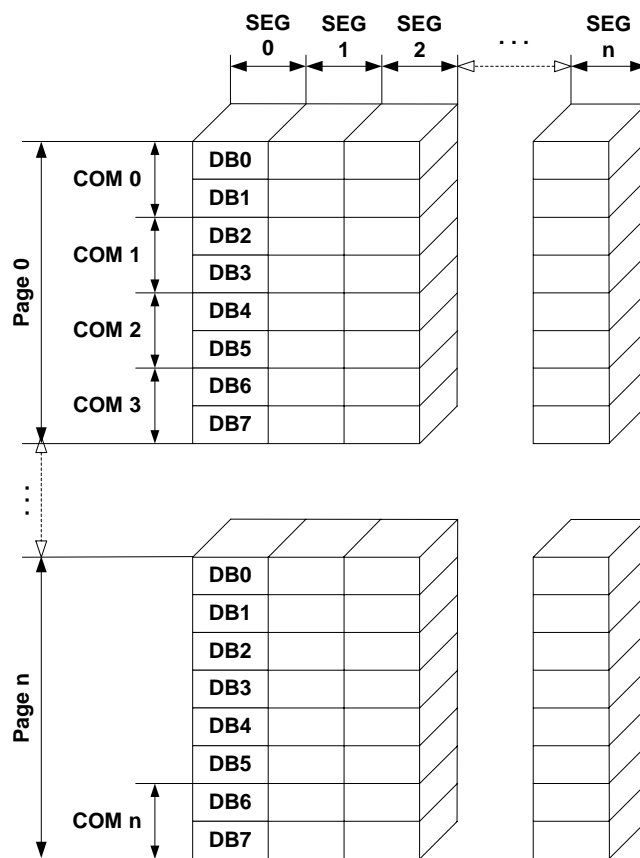
### Bits per pixel

Supported color depth is 2 bpp.

### Interfaces

Both 8-bit parallel (simple bus) and serial (SPI) interfaces are supported.

## Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

## Additional RAM requirements of the driver

This LCD driver may be used with or without a display data cache, containing a complete copy of the contents of the LCD data RAM. If a cache is not used, there are no additional RAM requirements.

It is recommended to use this driver with a data cache for faster LCD-access. The amount of memory used by the cache may be calculated as follows:

Size of RAM (in bytes) = (LCD\_YSIZE + 7) / 8 \* LCD\_XSIZE

## Additional driver functions

None.

## Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 20: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<a href="#">LCD_INIT_CONTROLLER</a>	Initialization sequence for the LCD controller.
<a href="#">LCD_READ_A0</a>	Read a byte from LCD controller with A-line low.
<a href="#">LCD_READ_A1</a>	Read a byte from LCD controller with A-line high.
<a href="#">LCD_WRITE_A0</a>	Write a byte to LCD controller with A-line low.
<a href="#">LCD_WRITE_A1</a>	Write a byte to LCD controller with A-line high.

## Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
<a href="#">LCD_CACHE</a>	When set to 0, no display data cache is used, which slows down the speed of the driver. Default is 1 (cache activated).

## Special requirements for certain LCD controllers

None.

# 22.6 LCD15XX

## Supported hardware

### Controllers

This driver has been tested with the following LCD controllers:

- Samsung KS0713
- Epson SED1560
- Epson SED1565
- Epson SED1566
- Epson SED1567
- Epson SED1568
- Epson SED1569
- Epson SED1575

It should be assumed that it will also work with any controller of similar organization.

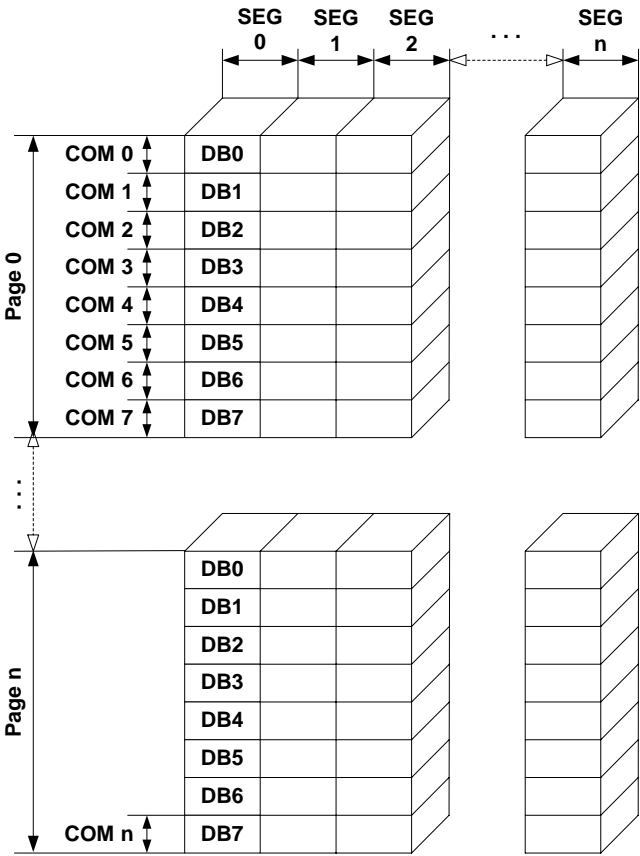
### Bits per pixel

Supported color depth is 1 bpp.

### Interfaces

Both 8-bit parallel (simple bus) and serial (SPI) interfaces are supported.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements of the driver

This LCD driver may be used with or without a display data cache, containing a complete copy of the contents of the LCD data RAM. If a cache is not used, there are no additional RAM requirements.

It is recommended to use this driver with a data cache for faster LCD-access. The amount of memory used by the cache may be calculated as follows:

Size of RAM (in bytes) = (LCD\_YSIZE + 7) / 8 \* LCD\_XSIZE

Additional driver functions

LCD\_L0\_ControlCache

For information about this function, please refer to Chapter 23: "LCD Driver API".

## Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 20: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<a href="#">LCD_INIT_CONTROLLER</a>	Initialization sequence for the LCD controller.
<a href="#">LCD_READ_A0</a>	Read a byte from LCD controller with A-line low.
<a href="#">LCD_READ_A1</a>	Read a byte from LCD controller with A-line high.
<a href="#">LCD_WRITE_A0</a>	Write a byte to LCD controller with A-line low.
<a href="#">LCD_WRITE_A1</a>	Write a byte to LCD controller with A-line high.

## Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
<a href="#">LCD_CACHE</a>	When set to 0, no display data cache is used, which slows down the speed of the driver. Default is 1 (cache activated).
<a href="#">LCD_SUPPORT_CACHECONTROL</a>	When set to 0, the cache control functions of <code>LCD_L0_ControlCache()</code> driver API are disabled.

## Special requirements for certain LCD controllers

None.

# 22.7 LCD6642X

## Supported hardware

### Controllers

This driver has been tested with the following LCD controllers:

- Hitachi HD66420
- Hitachi HD66421

It should be assumed that it will also work with any controller of similar organization.

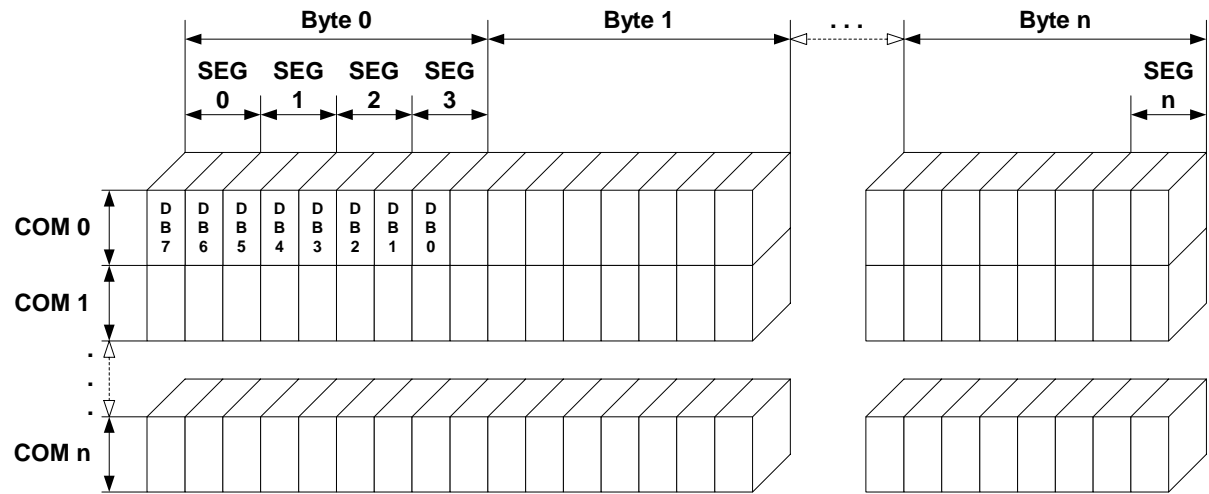
### Bits per pixel

Supported color depth is 2 bpp.

### Interfaces

The driver supports 8-bit parallel (simple bus) interfaces.

Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

Additional RAM requirements of the driver

This LCD driver may be used with or without a display data cache, containing a complete copy of the contents of the LCD data RAM. If a cache is not used, there are no additional RAM requirements. It is optional (but recommended) to use this driver with a data cache for faster LCD-access. The amount of memory used by the cache may be calculated as follows:  
Size of RAM (in bytes) = (LCD\_XSIZE + 7) / 8 \* LCD\_YSIZE \* 2

Additional driver functions

None.

Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 20: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<a href="#">LCD_INIT_CONTROLLER</a>	Initialization sequence for the LCD controller.
<a href="#">LCD_READ_A0</a>	Read a byte from LCD controller with A-line low.
<a href="#">LCD_READ_A1</a>	Read a byte from LCD controller with A-line high.
<a href="#">LCD_WRITE_A0</a>	Write a byte to LCD controller with A-line low.
<a href="#">LCD_WRITE_A1</a>	Write a byte to LCD controller with A-line high.

Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
<a href="#">LCD_CACHE</a>	When set to 0, no display data cache is used, which slows down the speed of the driver. Default is 1 (cache activated).

## Special requirements for certain LCD controllers

None.

## 22.8 LCDMem

### Using the CPU as LCD controller

In systems with relatively fast CPUs and small (quarter VGA or less) LCDs, there is no need for an LCD controller. The microcontroller (CPU) can do the job of the LCD controller on the side, refreshing the display in an interrupt service routine. The CPU's memory is used as video memory.

Advantages of this approach include the following:

- Very fast update of display possible.
- Eliminating the LCD controller (and its external RAM) reduces hardware costs.
- Simplified hardware design.
- 4 levels of gray can be displayed.

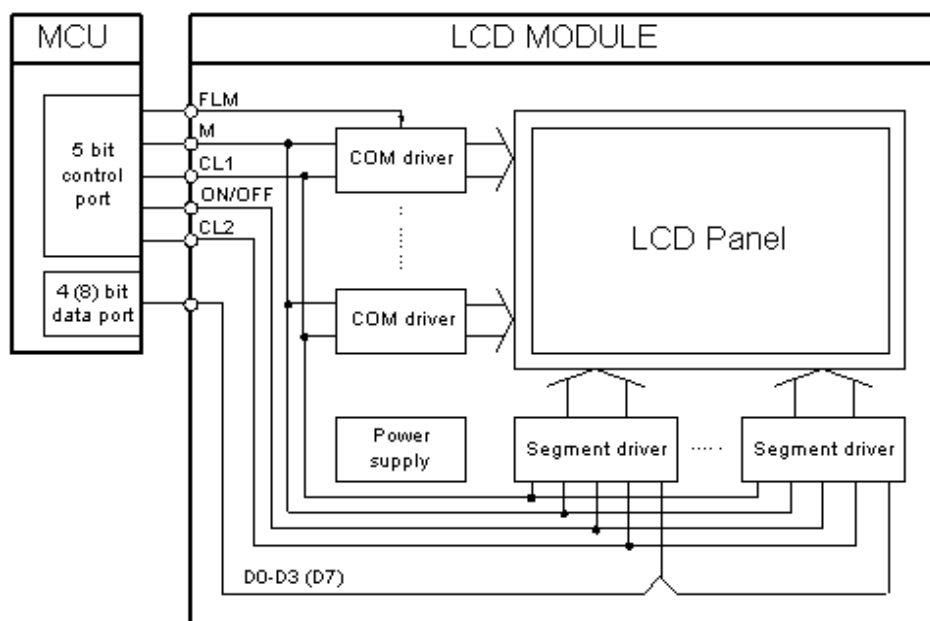
The disadvantage is that much of the available computation time is used up. Depending on the CPU, this can be anything between 20 and almost 100 percent; with slower CPUs, it is really not possible at all.

This type of interface does not require a specific LCD driver because  $\mu$ C/GUI simply places all the display data into the LCD cache. You yourself must write the hardware-dependent portion that periodically transfers the data in the cache memory to your LCD. Sample code for transferring the video image into the display is available in both "C" and optimized assembler for M16C and M16C/80.

### How to connect the CPU to the row/column drivers

It is quite easy to connect the microcontroller to the row/column drivers. Five control lines are needed, as well as either 4 or 8 data lines (depending on whether the column drivers are able to operate in 8-bit mode). 8-bit mode is recommended as it is

more efficient, saving calculation time of the CPU. All data lines should be on a single port, using port bits 0..3 or 0..7 in order to guarantee efficient access. This setup is illustrated below:



### CPU load

The CPU load depends on the hardware and controller used, as well as on the size of the display. For example:

Mitsubishi M16C62 Controller, 16MHz, 160\*100 display, 8-bit interface, 80 Hz update  
= app. 12% CPU load.

Mitsubishi M16C62 Controller, 16MHz, 240\*128 display, 8-bit interface, 80 Hz update  
= app. 22% CPU load.

## Supported hardware

### Controllers

None.

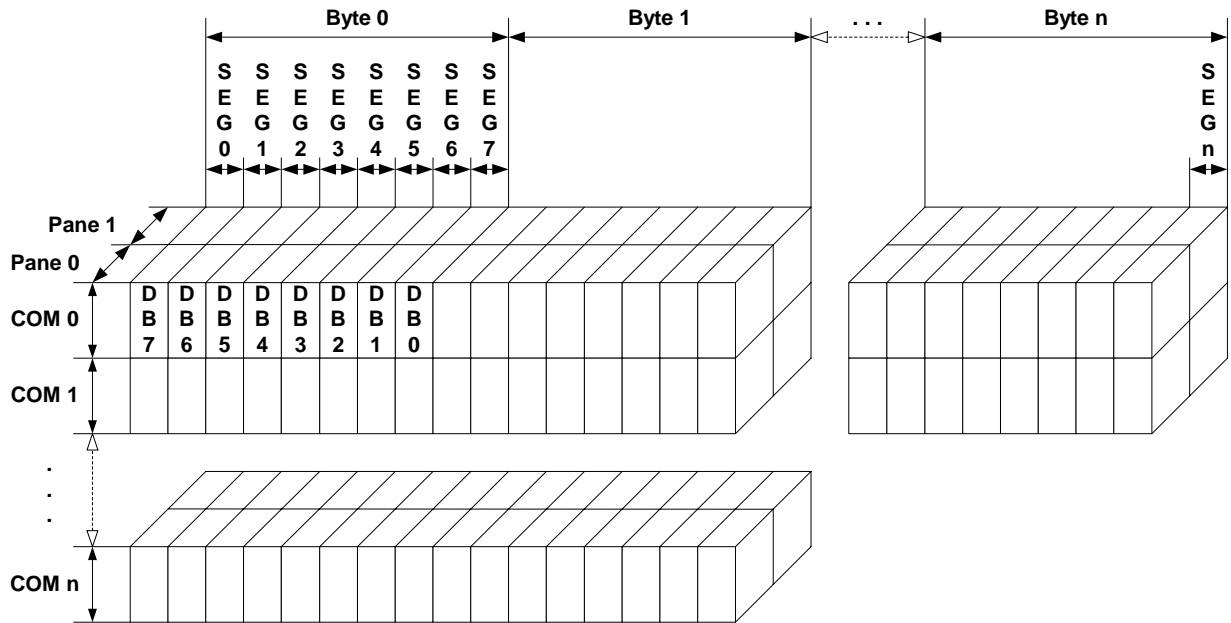
### Bits per pixel

Supported color depth is 2 bpp.

### Interfaces

The driver supports 1/4/8-bit interfaces from the CPU to the LCD.

## Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD. The display memory is divided into two panes for each pixel. The lower bit of one pixel is stored in pane 0 and the higher bit is stored in pane 1. The advantage of this method is that the output of the display data can be executed very quickly.

## RAM requirements of the driver

The driver only handles a memory area containing the display data. The required size of the display memory area may be calculated as follows:

Size of RAM (in bytes) = (LCD\_XSIZE + 7) / 8 \* LCD\_YSIZE \* 2

## Additional driver functions

None.

## Hardware configuration

Normally, the hardware interface is an interrupt service routine (ISR) which updates the LCD. An output routine written in "C" code is shipped with  $\mu$ C/GUI. This routine should serve only as an example. To optimize the execution speed, it must be adapted in assembler code.

For detailed information on how to write the output routine, please take a look at the sample supplied with the driver or contact us.

## Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
<code>LCD_TIMERINIT0</code>	Timing value used by ISR for displaying pane 0.

Macro	Explanation
<a href="#">LCD_TIMERINIT1</a>	Timing value used by ISR for displaying pane 1.
<a href="#">LCD_ON</a>	Function replacement macro which switches the LCD on.
<a href="#">LCD_OFF</a>	Function replacement macro which switches the LCD off.

## 22.9 LCDMemC

This driver, like LCDMem, is designed for a system without an LCD controller. The difference is that LCDMemC supports color displays. For more information on using the CPU instead of an LCD controller, please see the previous section on the LCDMem driver.

### Supported hardware

#### Controllers

None.

#### Bits per pixel

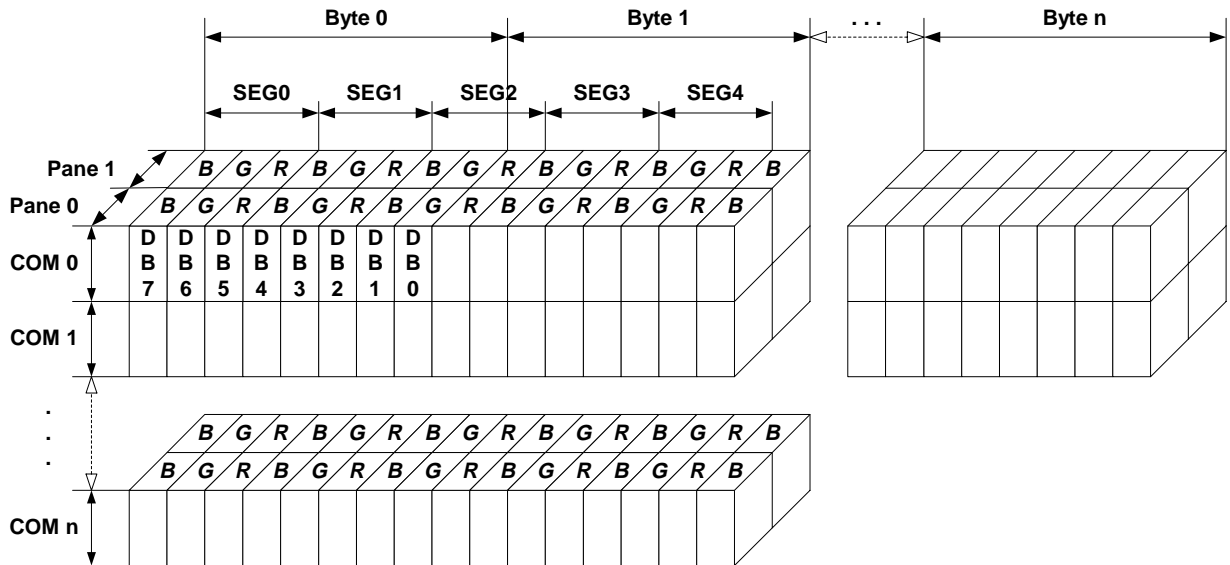
Supported color depths are 3 and 6 bpp.

#### Interfaces

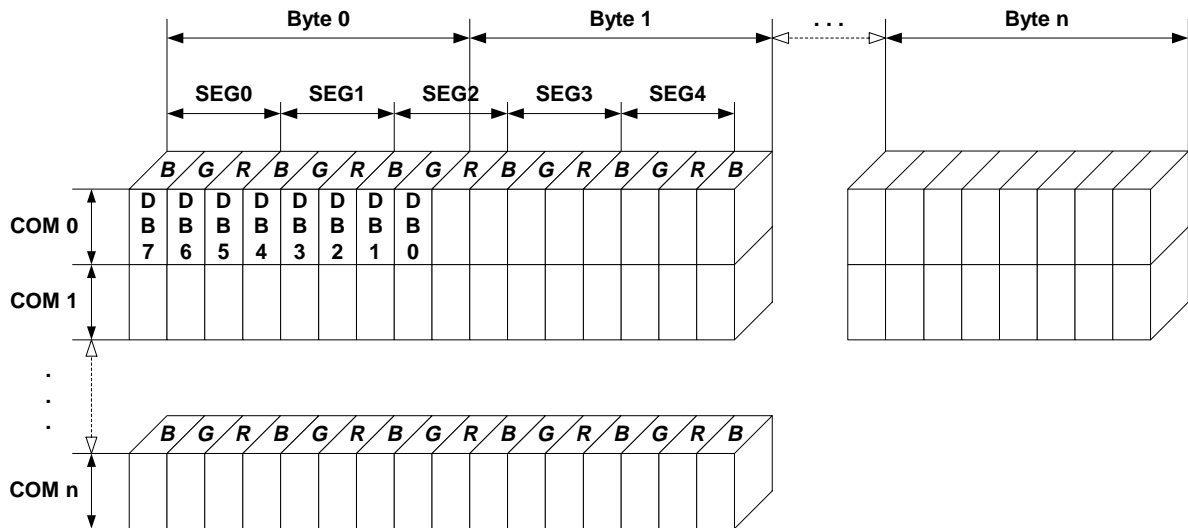
The driver supports 1/4/8-bit interfaces from the CPU to the LCD.

## Display data RAM organization

### 6 bits per pixel, fixed palette = 222



### 3 bits per pixel, fixed palette = 111



This driver supports a 3 or 6 bpp memory area for color displays. The pictures above show the dependence between the memory area handled by the driver and the SEG and COM lines of the LCD.

### 6 bits per pixel, fixed palette mode 222

When using the 6 bpp mode, the display memory is divided into 2 panes for each pixel. The lower bit of each pixel is stored in pane 0 and the higher bit is stored in pane 1. The advantage of this method is that the output of the display data can be executed very quickly.

**3 bits per pixel, fixed palette mode 111**  
When using this mode, only one pane exists for each pixel.

**RAM requirements of the driver**

The driver only handles a memory area containing the display data. The required size of the display memory area may be calculated as follows:

**6 bits per pixel, fixed palette mode 222**  
Size of RAM (in bytes) = (LCD\_XSIZE + 7) / 8 \* 3 \* 2

**3 bits per pixel, fixed palette mode 111**  
Size of RAM (in bytes) = (LCD\_XSIZE + 7) / 8 \* 3

**Additional driver functions**

None.

**Hardware configuration**

Normally, the hardware interface is an interrupt service routine (ISR) which updates the LCD. An output routine written in "C" code is shipped with µC/GUI. This routine should serve only as an example. To optimize the execution speed, it must be adapted in assembler code.  
For detailed information on how to write the output routine, please take a look at the sample supplied with the driver or contact us.

**Additional configuration switches**

The following table shows optional configuration switches available for this driver:

Macro	Explanation
<a href="#">LCD_TIMERINIT0</a>	Timing value used by ISR for displaying pane 0.
<a href="#">LCD_TIMERINIT1</a>	Timing value used by ISR for displaying pane 1 (only used by 6 bpp mode).
<a href="#">LCD_ON</a>	Function replacement macro which switches the LCD on.
<a href="#">LCD_OFF</a>	Function replacement macro which switches the LCD off.

**22.10 LCDPage1bpp**

**Supported hardware**

**Controllers**  
This driver has been tested with the following LCD controllers:

- Philips PCF8810
- Philips PCF8811

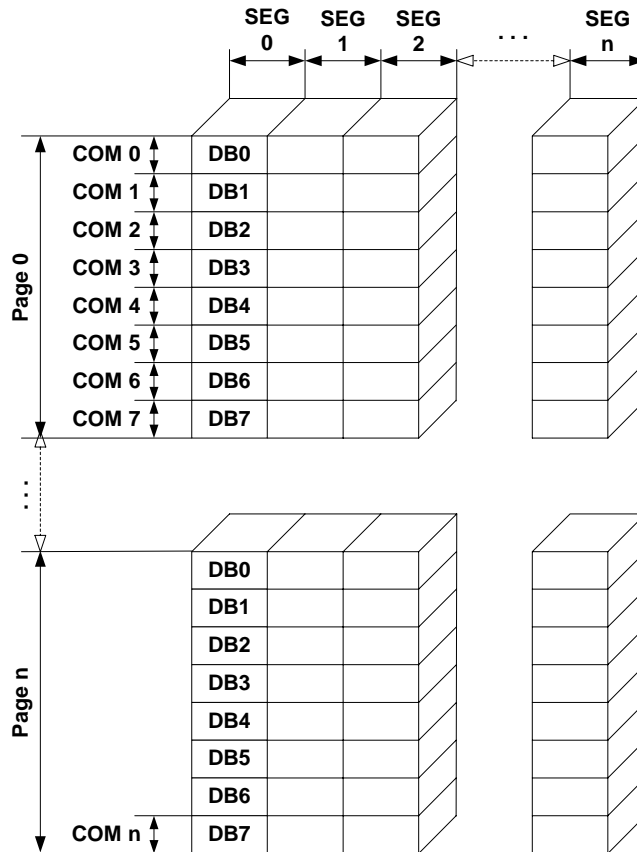
It should be assumed that it will also work with any controller of similar organization.

**Bits per pixel**  
Supported color depth is 1bpp.

## Interfaces

Both 8-bit parallel (simple bus) and serial (SPI) interfaces are supported.

## Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

## Additional RAM requirements of the driver

This LCD driver may be used with or without a display data cache, containing a complete copy of the contents of the LCD data RAM. If a cache is not used, there are no additional RAM requirements.

It is recommended to use this driver with a data cache for faster LCD-access. The amount of memory used by the cache may be calculated as follows:

Size of RAM (in bytes) =  $(\text{LCD\_YSIZE} + 7) / 8 * \text{LCD\_XSIZE}$

## Additional driver functions

### LCD\_L0\_ControlCache

For information about this function, please refer to Chapter 23: "LCD Driver API".

## Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 20: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<code>LCD_INIT_CONTROLLER</code>	Initialization sequence for the LCD controller.
<code>LCD_READ_A0</code>	Read a byte from LCD controller with A-line low.
<code>LCD_READ_A1</code>	Read a byte from LCD controller with A-line high.
<code>LCD_WRITE_A0</code>	Write a byte to LCD controller with A-line low.
<code>LCD_WRITE_A1</code>	Write a byte to LCD controller with A-line high.

## Additional configuration switches

The following table shows optional configuration switches available for this driver:

Macro	Explanation
<code>LCD_CACHE</code>	When set to 0, no display data cache is used, which slows down the speed of the driver. Default is 1 (cache activated).
<code>LCD_SUPPORT_CACHECONTROL</code>	When set to 0, the cache control functions of <code>LCD_L0_ControlCache()</code> driver API are disabled.

## Special requirements for certain LCD controllers

None.

# 22.11 LCDSLIn

## Supported hardware

### Controllers

This driver has been tested with the following LCD controllers:

- Epson SED1330
- Epson SED1335
- Toshiba T6963

It should be assumed that it will also work with any controller of similar organization.

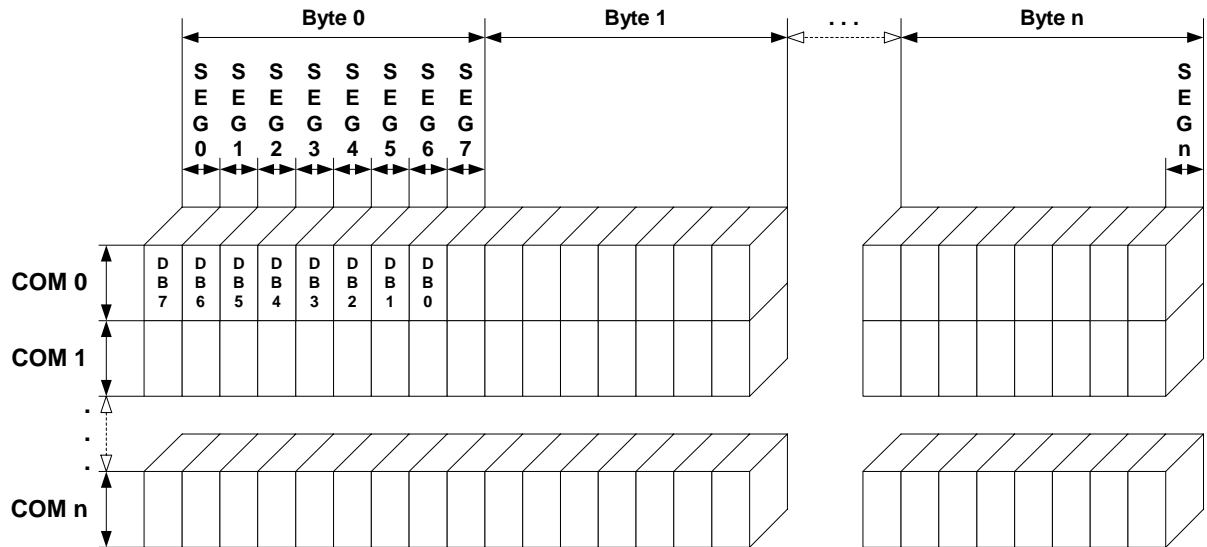
### Bits per pixel

Supported color depth is 1 bpp.

### Interfaces

The driver supports 8-bit parallel (simple bus) interfaces.

## Display data RAM organization



The picture above shows the relation between the display memory and the SEG and COM lines of the LCD.

## Additional RAM-requirement

This LCD driver may be used with or without a display data cache, containing a complete copy of the contents of the LCD data RAM. If a cache is not used, there are no additional RAM requirements.

It is recommended to use this driver with a data cache for faster LCD-access. The amount of memory used by the cache may be calculated as follows:

Size of RAM (in bytes) =  $(\text{LCD\_XSIZE} + 7) / 8 * \text{LCD\_YSIZE}$

## Additional driver functions

None.

## Hardware configuration

This driver accesses the hardware with a simple bus interface as described in Chapter 20: "Low-Level Configuration". The following table lists the macros which must be defined for hardware access:

Macro	Explanation
<a href="#">LCD_INIT_CONTROLLER</a>	Initialization sequence for the LCD controller.
<a href="#">LCD_READ_A0</a>	Read a byte from LCD controller with A-line low.
<a href="#">LCD_READ_A1</a>	Read a byte from LCD controller with A-line high.
<a href="#">LCD_WRITE_A0</a>	Write a byte to LCD controller with A-line low.
<a href="#">LCD_WRITE_A1</a>	Write a byte to LCD controller with A-line high.

## Additional configuration switches

None.

## **Special requirements for certain LCD controllers**

None.

# Chapter 23

## LCD Driver API

---

$\mu$ C/GUI requires a driver for the hardware. This chapter explains what an LCD driver for  $\mu$ C/GUI does and what routines it supplies to  $\mu$ C/GUI (the application program interface, or API).

Under most circumstances, you probably do not need to read this chapter, as most calls to the LCD layer of  $\mu$ C/GUI will be done through the GUI layer. In fact, we recommend that you only call LCD functions if there is no GUI equivalent (for example, if you wish to modify the lookup table of the LCD controller directly). The reason for this is that LCD driver functions are not thread-safe, unlike their GUI equivalents. They should therefore not be called directly in multitask environments.

## 23.1 LCD driver API

The table below lists the available  $\mu$ C/GUI LCD-related routines in alphabetical order. Detailed descriptions of the routines can be found in the sections that follow.

### LCD\_L0: Driver routines

Routine	Explanation
Init & display control group	
<a href="#">LCD_L0_Init()</a>	Initialize the display.
<a href="#">LCD_L0_ReInit()</a>	Reinitialize LCD without erasing the contents.
<a href="#">LCD_L0_Off()</a>	Switch LCD off.
<a href="#">LCD_L0_On()</a>	Switch LCD on.
Drawing group	
<a href="#">LCD_L0_DrawBitmap()</a>	Universal draw bitmap routine.
<a href="#">LCD_L0_DrawHLine()</a>	Draw a horizontal line.
<a href="#">LCD_L0_DrawPixel()</a>	Draw a pixel in the current foreground color.
<a href="#">LCD_L0_DrawVLine()</a>	Draw a vertical line.
<a href="#">LCD_L0_FillRect()</a>	Fill a rectangular area.
<a href="#">LCD_L0_SetPixelIndex()</a>	Draw a pixel in a specified color.
<a href="#">LCD_L0_XorPixel()</a>	Invert a pixel.
"Get" group	
<a href="#">LCD_L0_GetPixelIndex()</a>	Returns the index of the color of a specific pixel.
"Set" group	
<a href="#">LCD_L0_SetOrg()</a>	Not yet used, reserved for future use (must exist in driver).
Lookup table group	
<a href="#">LCD_L0_SetLUTEntry()</a>	Modify a single entry of LUT.
Misc. group (optional)	
<a href="#">LCD_L0_ControlCache()</a>	Lock/unlock/flush LCD cache.

## LCD: LCD layer routines

Routine	Explanation
<a href="#">LCD_GetXSize()</a>	Return physical X-size of LCD in pixels.
<a href="#">LCD_GetYSize()</a>	Return physical Y-size of LCD in pixels.
<a href="#">LCD_GetVXSize()</a>	Return virtual X-size of LCD in pixels.
<a href="#">LCD_GetVYSize()</a>	Return virtual Y-size of LCD in pixels.
<a href="#">LCD_GetBitsPerPixel()</a>	Return the number of bits per pixel.
<a href="#">LCD_GetNumColors()</a>	Return the number of available colors.
<a href="#">LCD_GetFixedPalette()</a>	Return the fixed palette mode.

## 23.2 Init & display control group

### LCD\_L0\_Init()

#### Description

Initializes the LCD using the configuration settings in `LCDConf.h`. This routine is called automatically by `GUI_Init()` if the upper GUI layer is used and therefore should not need to be called manually.

#### Prototype

```
void LCD_L0_Init (void);
```

### LCD\_L0\_ReInit()

#### Description

Reinitializes the LCD using the configuration settings, without erasing the display contents.

#### Prototype

```
void LCD_L0_ReInit (LCD_INITINFO* pInitInfo);
```

### LCD\_L0\_Off (), LCD\_L0\_On()

#### Description

Switch the display off or on, respectively.

#### Prototypes

```
void LCD_L0_Off(void);
void LCD_L0_On(void);
```

#### Additional information

Use of these routines does not affect the contents of the video memory or other settings. You may therefore safely switch off the display and switch it back on without having to refresh the contents.

## 23.3 Drawing group

### LCD\_L0\_DrawBitMap()

**Description**

Draws a pre-converted bitmap.

**Prototype**

```
LCD_L0_DrawBitMap(int x0, int y0,
                  int Xsize, int Ysize,
                  int BitsPerPixel,
                  int BytesPerLine,
                  const U8* pData, int Diff,
                  const LCD_PIXELINDEX* pTrans);
```

Parameter	Meaning
x0	Upper left X-position of bitmap to draw.
y0	Upper left Y-position of bitmap to draw.
Xsize	Number of pixels in horizontal direction.
Ysize	Number of pixels in vertical direction.
BitsPerPixel	Number of bits per pixel.
BytesPerLine	Number of bytes per line of the image.
pData	Pointer to the actual image, the data that defines what the bitmap looks like.
Diff	Number of pixels to skip from the left side.

### LCD\_L0\_DrawHLine()

**Description**

Draws a horizontal line one pixel thick, at a specified position using the current foreground color.

**Prototype**

```
void LCD_L0_DrawHLine(int x0, int y, int x1);
```

Parameter	Meaning
x0	Start position of line.
y	Y-position of line to draw.
x1	End position of line.

**Additionalnall information**

With most LCD controllers, this routine executes very quickly because multiple pixels can be set at once and no calculations are needed. If it is clear that horizontal lines are to be drawn, this routine executes faster than the DrawLine routine.

### LCD\_L0\_DrawPixel()

**Description**

Draws one pixel at a specified position using the current foreground color.

## Prototype

```
void LCD_L0_DrawPixel(int x, int y);
```

Parameter	Meaning
<a href="#">x</a>	X-position of pixel to draw.
<a href="#">y</a>	Y-position of pixel to draw.

## LCD\_L0\_DrawVLine()

### Description

Draws a vertical line one pixel thick, at a specified position using the current foreground color.

### Prototype

```
void LCD_L0_DrawVLine(int x , int y0, int y1);
```

Parameter	Meaning
<a href="#">x</a>	X-position of line to draw.
<a href="#">y0</a>	Start position of line.
<a href="#">y1</a>	End position of line.

### Additional information

With most LCD-controllers, this routine executes very quickly because multiple pixels can be set at once and no calculations are needed. If it is clear that horizontal lines are to be drawn, this routine executes faster than the `DrawLine` routine.

## LCD\_L0\_FillRect()

### Description

Draws a filled rectangle at a specified position using the current foreground color.

### Prototype

```
void LCD_L0_FillRect(int x0, int y0, int x1, int y1);
```

Parameter	Meaning
<a href="#">x0</a>	Upper left X-position.
<a href="#">y0</a>	Upper left Y-position.
<a href="#">x1</a>	Lower right X-position.
<a href="#">y1</a>	Lower right Y-position.

## LCD\_L0\_SetPixelIndex()

### Description

Draws one pixel using a specified color

## Prototype

```
void LCD_L0_SetPixelIndex(int x, int y, int ColorIndex);
```

Parameter	Meaning
<a href="#">x</a>	X-position of pixel to draw.
<a href="#">y</a>	Y-position of pixel to draw.
<a href="#">ColorIndex</a>	Color to be used.

## LCD\_L0\_XorPixel()

### Description

Inverts one pixel.

### Prototype

```
void LCD_L0_XorPixel(int x, int y);
```

Parameter	Meaning
<a href="#">x</a>	X-position of pixel to invert.
<a href="#">y</a>	Y-position of pixel to invert.

## 23.4 "Get" group

### LCD\_L0\_GetPixelIndex()

#### Description

Returns the RGB color index of a specified pixel.

#### Prototype

```
int LCD_L0_GetPixelIndex(int x, int y);
```

Parameter	Meaning
<a href="#">x</a>	X-position of pixel.
<a href="#">y</a>	Y-position of pixel.

#### Return value

The index of the pixel.

#### Additionnal information

For further information see Chapter 9: "Colors".

## 23.5 Lookup table (LUT) group

### LCD\_L0\_SetLUTEntry()

#### Description

Modifies a single entry to the LUT of the LCD controller(s).

## Prototype

```
void LCD_L0_SetLUTEntry(U8 Pos, LCD_COLOR Color;
```

Parameter	Meaning
<code>Pos</code>	Position within the lookup table. Should be less than the number of colors, e.g. 0-3 for 2bpp, 0-15 for 4bpp, 0-255 for 8bpp.
<code>Color</code>	24-bit RGB value. The closest value possible will be used for the LUT. If a color LUT is to be initialized, all 3 components are used. In monochrome modes the green component is used, but it is still recommended (for better understanding of the program code) to set all 3 colors to the same value (such as 0x555555 or 0xa0a0a0).

## 23.6 Miscellaneous group

### LCD\_L0\_ControlCache()

#### Description

Locks, unlocks or flushes the cache. This routine may be used to set the cache to a locked state, in which all drawing operations on the driver cause changes in the video memory's cache (in CPU RAM), but do not cause any visible output. Unlocking or flushing then causes those changes to be written to the display. This can help to avoid flickering of the display and also accelerate drawing. It does not matter how many different drawing operations are executed; the changes will all be written to the display at once. In order to be able to do this, `LCD_SUPPORT_CACHECONTROL` must be enabled in the configuration file.

#### Prototype

```
U8 LCD_ControlCache(U8 command);
```

Parameter	Meaning
<code>command</code>	Specify the command to be given to the cache. Use the symbolic values in the table below.

Permitted values for parameter <code>command</code>	
<code>LCD_CC_UNLOCK</code>	Set the default mode: cache is transparent.
<code>LCD_CC_LOCK</code>	Lock the cache, no write operations will be performed until cache is unlocked or flushed.
<code>LCD_CC_FLUSH</code>	Flush the cache, writing all modified data to the video RAM.

#### Return value

Information on the state of the cache. Ignore.

#### Additional information

When the cache is locked, the driver maintains a "hitlist" -- a list of bytes which have been modified and need to be written to the display. This hitlist uses 1 bit per byte of video memory.

This is an optional feature which is not supported by all LCD drivers

#### Example

The code in the following example performs drawing operations on the display which overlap. In order to accelerate the update of the display and to avoid flickering, the cache is locked before and unlocked after these operations.

```
LCD_ControlCache(LCD_CC_LOCK);

GUI_FillCircle(30,30,20);
GUI_SetDrawMode(GUI_DRAWMODE_XOR);
GUI_FillCircle(50,30,10);
GUI_SetTextMode(GUI_TEXTMODE_XOR);
GUI_DispStringAt("Hello world\n",0,0);
GUI_DrawHLine(16, 5,25);
GUI_DrawHLine(18, 5,25);
GUI_DispStringAt("XOR Text",0,20);
GUI_DispStringAt("XOR Text",0,60);

LCD_ControlCache(LCD_CC_UNLOCK);
```

## **LCD\_GetXSize(), LCD\_GetYSize()**

### **Description**

Returns the physical X- or Y-size, respectively, of the LCD in pixels.

### **Prototypes**

```
int LCD_GetXSize(void)
int LCD_GetYSize(void)
```

### **Return value**

Physical X/Y-size of the display.

## **LCD\_GetVXSize(), LCD\_GetVYSize()**

### **Description**

Returns the virtual X- or Y-size, respectively, of the LCD in pixels. In most cases, the virtual size is equal to the physical size.

### **Prototype**

```
int LCD_GetVXSize(void)
int LCD_GetVYSize(void)
```

### **Return value**

Virtual X/Y-size of the display.

## **LCD\_GetBitsPerPixel()**

### **Description**

Returns the number of bits per pixel.

### **Prototype**

```
int LCD_GetBitsPerPixel(void);
```

### **Return value**

Number of bits per pixel.

## **LCD\_GetNumColors()**

### **Description**

Returns the number of currently available colors on the LCD.

**Prototype**

```
int LCD_GetNumColors(void);
```

**Return value**

Number of available colors

**LCD\_GetFixedPalette()****Description**

Returns the fixed palette mode.

**Prototype**

```
int LCD_GetFixedPalette(void);
```

**Return value**

The fixed palette mode. See Chapter 9: "Colors" for more information on fixed palette modes.



# Chapter 24

## Performance and Resource Usage

---

High performance combined with low resource usage has always been a major design consideration.  $\mu$ C/GUI runs on 8/16/32-bit CPUs. Depending on which modules are being used, even single-chip systems with less than 64kb ROM and 2kb RAM can be supported by  $\mu$ C/GUI. The actual performance and resource usage depends on many factors (CPU, compiler, memory model, optimization, configuration, interface to LCD controller, etc.). This chapter contains benchmarks and information about resource usage in typical systems which can be used to obtain sufficient estimates for most target systems.

## 24.1 Performance benchmark

We use a benchmark test to measure the speed of the software on available targets. This benchmark is in no way complete, but it gives an approximation of the length of time required for common operations on various targets.

### Configuration and performance table (all values are in $\mu\text{s}/\text{pixel}$ )

CPU	LCD Controller	bpp	Bench1 Filling	Bench2 Small fonts	Bench3 Big fonts	Bench4 Bitmap 1bpp	Bench5 Bitmap 2bpp	Bench6 Bitmap 4bpp	Bench7 Bitmap 8bpp	Bench8 DDP bitmap
M16C/60 (16 bit), 16MHz	T6963	1								
M16C/60 (16 bit), 16MHz	T6963	1								
M16C/80 (16 bit), 20MHz	1375	1	0.26	9.26	4.43	7.38	7.93	8.01	7.99	7.14
M16C/80 (16 bit), 20MHz	1375	4	0.46	5.60	2.29	2.94	8.21	3.14	7.86	1.54
M16C/80 (16 bit), 20MHz	1375	8	0.63	5.45	2.30	3.26	7.65	3.23	2.81	1.61

### Description of test sequences used in the benchmark test

#### Bench1: Filling

Bench the speed of filling. An area of 64\*64 pixels is filled with different colors.

#### Bench2: Small fonts

Bench the speed of small character output. An area of 60\*64 pixels is filled with small-character text.

#### Bench3: Big fonts

Bench the speed of big character output. An area of 65\*48 pixels is filled with big-character text.

#### Bench4: Bitmap 1bpp

Bench the speed of 1bpp bitmaps. An area of 58\*8 pixels is filled with a 1bpp bitmap.

#### Bench 5: Bitmap 2bpp

Bench the speed of 2bpp bitmaps. An area of 32\*11 pixels is filled with a 2bpp bit-map.

#### Bench6: Bitmap 4bpp

Bench the speed of 4bpp bitmaps. An area of 32\*11 pixels is filled with a 4bpp bit-map.

#### Bench7: Bitmap 8bpp

Bench the speed of 8bpp bitmaps. An area of 32\*11 pixels is filled with a 8bpp bit-map.

## Bench8: Device-dependent bitmap, 8 or 16 bpp

Bench the speed of bitmaps 8 or 16 bits per pixel. An area of 64\*8 pixels is filled with a bitmap. The color depth of the tested bitmap depends on the configuration. For configurations  $\leq 8$ bpp, a bitmap with 8 bpp is used; 16bpp configurations use a 16-bpp bitmap.

## 24.2 Memory requirements

The following table will give you an idea of the memory requirements for  $\mu$ C/GUI. The values in this table are approximations depending on the CPU, the C compiler used, the memory model and the compile time switches (i.e. which parts of  $\mu$ C/GUI are included in your application). However, the following data represents a good average value for 32-bit CPUs, based on the requirements for x86 and Fujitsu FR30 CPUs. For 16-bit CPUs, the ROM code size should be smaller (app. 30% in native memory model, i.e. with 16-bit pointers); the ROM size for data (fonts) is identical.

Short description	Switch	RAM [bytes]	ROM [bytes]
<b>Basic system</b>			
Core software, without fonts and graphic library.		120	3900
<b>Color (palette) management</b>			
Color management, support for 16 colors, no cache.		16	336
Color management, support for 256 colors, no cache.		256	336
Color management, support for 256 colors, 1000 byte caching.		1256	580
<b>Fonts</b>			
F4x6 font (ASCII only).		---	600
F6x8 font, (ISO8859-1).		---	1536
F8x8 font, (ISO8859-1).		---	1536
F8x16 font, (ISO8859-1).		---	3072
FD24x32 font (big digits).		---	1374
F4x6 font (ASCII only).		---	600
<b>Graphic library</b>			
Bitmaps.		---	u.i.
Simple lines, any angle.		---	u.i.
Lines with line styles.		---	u.i.
Polylines, drawing.		---	u.i.
Polylines, filling (polygons).		---	u.i.
Circles, drawing.		---	u.i.
Circles, filling.		---	u.i.
***** Total		---	9500
<b>Memory devices (optional)</b>			
Core software		depends on area size	ca. 500
Support for 1 bit/pixel drivers.		---	ca. 1800
Support for 2 bit/pixel drivers.		---	ca. 2400
Support for 4 bit/pixel drivers.		---	ca. 2400
Support for 8 bit/pixel drivers.		---	ca. 1700
<b>Graphic library, antialiased (u.d.)</b>			
Simple lines, any angle.			u.i.

Short description	Switch	RAM [bytes]	ROM [bytes]
Polygons.			u.i.
Circles.			u.i.
Driver			
SED1565, single LCD controller, core.		12	1600
SED1565, additionnal support for palette management.			ca. 100
SED1565, additionnal support for 4-color bitmaps for driver.			ca. 800
SED1565, additionnal support for 16-color bitmaps for driver.			ca. 600
SED1565, additionnal support for 256-color bitmaps for driver.			ca. 520
SED1565, additionnal support for scaled bitmaps for driver.			ca. 260
SED1565, additionnal support for antialiasing for driver.			ca. 80
SED1352, single LCD controller, core.			
SED1352, dual LCD controller, core.			
Window manager			
Core software.			
Additional window.		ca. 40	---

# Chapter 25

## Standard Fonts

---

µC/GUI is shipped with various fonts which should cover most of your needs. The standard font package contains monospaced and proportional fonts in different sizes and styles. This chapter provides an overview of these fonts.

### **Monospaced fonts**

Fonts with a fixed character width. All characters have the same width in pixels.

### **Proportional fonts**

Each character has an individual width in pixels.

This chapter provides an overview of the standard µC/GUI fonts.

## 25.1 Font identifier naming convention

All standard fonts are named as follows. The elements of the naming convention are then explained in the table:

GUI\_Font[<style>][<width>x]<height>[x<MagX>x<MagY>][H][B][\_<characterset>]

Element	Meaning
GUI_Font	Standard prefix for all fonts shipped with µC/GUI.
<style>	Specifies a non-standard font style. Example: Comic style in GUI_FontComic18B_ASCII.
<width>	Width of characters, contained only in monospaced fonts.
<height>	Height of the font in pixels.
<MagX>	Factor of magnification in X, contained only in magnified fonts.
<MagY>	Factor of magnification in Y, contained only in magnified fonts.
H	Abbreviation for "high". Only used if there is more than one font with the same height. It means that the font appears "higher" than other fonts.
B	Abbreviation for "bold". Used in bold fonts.
<characterset>	Specifies the contents of characters: ASCII: Only ASCII characters 0x20-0x7E (0x7F). 1: ASCII characters and European extensions 0xA0 - 0xFF. HK: Hiragana and Katakana. 1HK: ASCII, European extensions, Hiragana and Katakana. D: Digit fonts, character set: +-.0123456789.

### Example 1:

GUI\_Font16\_ASCII

Element	Meaning
GUI_Font	Standard font prefix.
16	Height in pixels.
ASCII	Font contains ASCII characters only.

### Example 2:

GUI\_Font8x15B\_ASCII

Element	Meaning
GUI_Font	Standard font prefix.
8	Width of characters.
x15	Height in pixels.
B	Bold font.
_ASCII	Font contains ASCII characters only.

**Example 3:**

GUI\_Font8x16x1x2

Element	Meaning
GUI_Font	Standard font prefix.
8	Width of characters.
x16	Height in pixels.
x1	Magnification factor in X.
x2	Magnification factor in Y.

## 25.2 Font file naming convention

The names for the font files are similar to the names of the fonts themselves. The files are named as follows:

F[<width>]<height>[H][B][<characterset>]

Element	Meaning
F	Standard prefix for all fonts files shipped with $\mu$ C/GUI.
<width>	Width of characters, contained only in monospaced fonts.
<height>	Height of the font in pixels.
H	Abbreviation for "high". Only used if there is more than one font with the same height. It means that the font appears "higher" than other fonts.
B	Abbreviation for "bold". Used in bold fonts.
<characterset>	Specifies the contents of characters: ASCII: Only ASCII characters 0x20-0x7E (0x7F). 1: ASCII characters and European extensions 0xA0 - 0xFF. HK: Hiragana and Katakana. 1HK: ASCII, European extensions, Hiragana and Katakana. D: Digit fonts.

## 25.3 Measurement, ROM-size and character set of fonts

The following pages describe the standard fonts shipped with  $\mu$ C/GUI. There is a measurement diagram, an overview of all characters included and a table containing the ROM size in bytes and the font files required when using the font.

The following parameters are used in the measurement diagrams:

Element	Meaning
F	Size of font in Y.
B	Distance of base line from the top of the font.
C	Height of capital characters.
L	Height of lowercase characters.
U	Size of underlength used by letters such as "g", "j" or "y".



ROM Size	1800 bytes
Used files	F10_ASCII.c

GUI\_Font10\_1

F: 10  
B: 09  
C: 08  
L: 06  
U: 01



!"#\$%&'()\*+,-./0123456789;<=>?@ABCDEFGHIJKLMNPOQRSTUVWXYZ  
WXYZ[^\\_`abcdefghijklmnopqrstuvwxyz{|}~ ¡¢£¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ

ROM Size	2456 bytes + size of (GUI_Font10_ASCII)
Used files	F10_1.c, F10_ASCII.c

GUI\_Font10S\_ASCII

F: 10  
B: 08  
C: 06  
L: 04  
U: 02



!"#\$%&'()\*+,-./0123456789;<=>?@ABCDEFGHIJKLMNPOQRSTUVWXYZ[^\\_`abcde  
fghijklmnopqrstuvwxyz{|}~

ROM Size	1760 bytes
Used files	F10_ASCII.c

GUI\_Font10S\_1

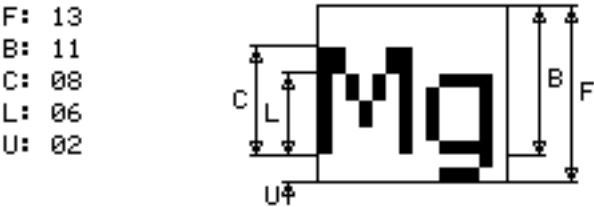
F: 10  
B: 08  
C: 06  
L: 04  
U: 02



!"#\$%&'()\*+,-./0123456789;<=>?@ABCDEFGHIJKLMNPOQRSTUVWXYZ[\]^\_`abcde  
fghijklmnopqrstuvwxyz{|}~ ¡¢£¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓ  
ÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ

ROM Size	1770 bytes + size of (GUI_Font10S_ASCII)
Used files	F10S_1.c, F10_ASCII.c

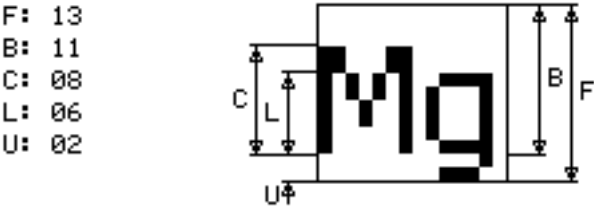
GUI\_Font13\_ASCII



!"#\$%&'()\*+,-./0123456789;<=>?@ABCDEFGHIJKLMNPOQRS  
TUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~

ROM Size	2076 bytes
Used files	F13_ASCII.c

GUI\_Font13\_1



!"#\$%&'()\*+,-./0123456789;<=>?@ABCDEFGHIJKLMNPOQRS  
TUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~ ¡¢£¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ

ROM Size	2149 bytes + size of (GUI_Font13_ASCII)
Used files	F13_1.c, F13_ASCII.c

GUI\_Font13H\_ASCII

F: 13  
B: 11  
C: 09  
L: 07  
U: 02



!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLM  
NOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{  
|}~

ROM Size	2232 bytes
Used files	F13H_ASCII.c

GUI\_Font13H\_1

F: 13  
B: 11  
C: 09  
L: 07  
U: 02



!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLM  
NOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{  
|}~ ¡¢£¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆ  
ÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïð  
ñôõ÷øùúûüýþ

ROM Size	2291bytes + size of (GUI_Font13H_ASCII)
Used files	F13H_1.c, F13H_ASCII.c

GUI\_Font13HB\_ASCII

F: 13  
B: 11  
C: 09  
L: 07  
U: 02



!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEFGH  
IJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz  
rstuvwxyz{|}~

ROM Size	2690 bytes
Used files	F13HB_ASCII.c

GUI\_Font13HB\_1

F: 13  
B: 11  
C: 09  
L: 07  
U: 02



!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEFGH  
IJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz  
rstuvwxyz{|}~ ¡¢£¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»  
¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞ  
ßàáâãäåæçèéêëìíîïðñóôõö÷øùúûüýþ

ROM Size	2806 bytes + size of (GUI_Font13HB_ASCII)
Used files	F13HB_1.c, F13HB_ASCII.c

GUI\_Font16\_ASCII

F: 16  
B: 13  
C: 10  
L: 07  
U: 03



!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLM  
NOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{  
|}~

ROM Size	2714 bytes
Used files	F16_ASCII.c

### GUI\_Font16\_1

F: 16  
B: 13  
C: 10  
L: 07  
U: 03



!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLM  
NOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{  
|}~ ¡¢£¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊ  
ËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñóôõ  
ö÷øùúûüýþ

ROM Size	3850 bytes + size of (GUI_Font16_ASCII)
Used files	F16_1.c, F16_ASCII.c

### GUI\_Font16\_HK

ああいいううええおおかがきぎくぐけげこご  
さざしじすずせぜそぞただちぢっつづてでと  
どなにぬねのはばびひびふぶへべへほほ  
ぼまみむめもやゆよよりるれろわわゐ  
ゑをんァアイイウウエエォオカガキギクグケ  
ゲコゴサザシジスズセゼソゾタダチヂッツヅ  
テデトドナニヌネノハババヒビビフブフヘベ  
ペホポボマミムメモャヤユョヨラリルレロ  
ワヰヱヰンヴカケ

ROM Size	6950 bytes
Used files	F16_HK.c

GUI\_Font16\_1HK

F: 16

B: 13

C: 10

L: 07

U: 03

!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLM  
NOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|  
}~ ¡¢£¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊË  
ÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãääåæçèéêëìíîïðñòóôõ  
ö÷øùúûüýþÿ ああいうえお おかがきぎくく  
けげごさざしじすずせぜそぞただちぢっつ  
づてでとどなにぬねのはばぱひびぴふぶぷ  
べへほぼまみむめもやゃゆゅょよらりるれ  
ろわわゐゑをんァアィイウウェエォオカガキ  
ギクグケゲコゴサザシジスズセゼソゾタダチ  
ヂッツヅテデトドナニヌネノハバパヒビピフ  
ブプヘベペホボボマミムメモャヤユヨョラ  
リルレロワウヰエヲンヴカケ

ROM Size	120 bytes + size of (GUI_Font16_HK) + size of (GUI_Font16_1) + size of (GUI_Font16_ASCII)
Used files	F16_1HK.c, F16_HK.c, F16_1.c, F16_ASCII.c

GUI\_Font16B\_ASCII

F: 16

B: 13

C: 10

L: 07

U: 03

!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLM  
NOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstu  
vwxyz{|}~

ROM Size	2690 bytes
Used files	F16B_ASCII.c

GUI\_Font16B\_1

F: 16  
B: 13  
C: 10  
L: 07  
U: 03



!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLM  
NOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz  
xyz{|}~ ¡¢£¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆ  
ÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìí  
îïðñòóôõö÷øùúûüýþ

ROM Size	2790 bytes + size of (GUI_Font16B_ASCII)
Used files	F16B_1.c, F16B_ASCII.c

GUI\_FontComic18B\_ASCII

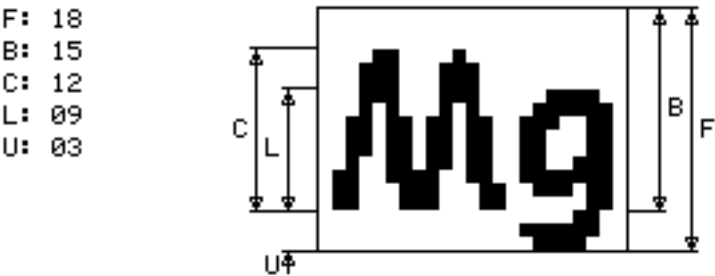
F: 18  
B: 15  
C: 12  
L: 09  
U: 03



!"#\$%&'()\*+,-./0123456789:;<=>?@ABCD  
EFGHIJKLMNOPQRSTUVWXYZ[\]^\_`ab  
cdefghijklmnopqrstuvwxyz{|}~€

ROM Size	3572 bytes
Used files	FComic18B_ASCII.c

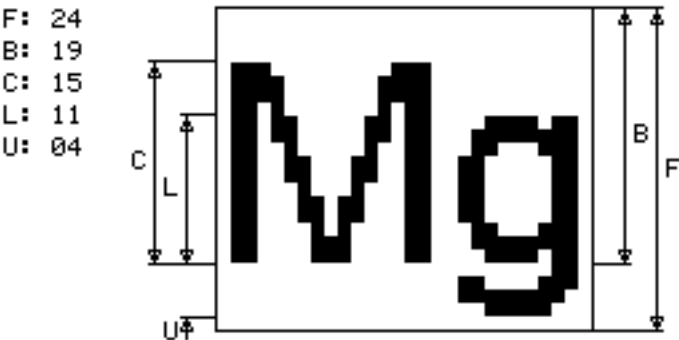
GUI\_FontComic18B\_1



!"#\$%&'()\*+,-./0123456789:;<=>?@ABCD  
EFGHIJKLMNOPQRSTUVWXYZ[\]^\_`ab  
cdefghijklmnopqrstuvwxyz{|}~ ¡¢£¥¦§¨ª«  
¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉ  
ÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæç  
èéêëìíîïðñòóôõö÷øùúûüýþ

ROM Size	4334 bytes + size of(GUI_FontComic18b_ASCII)
Used files	FComic18B_1c, FComic18B_ASCII.c

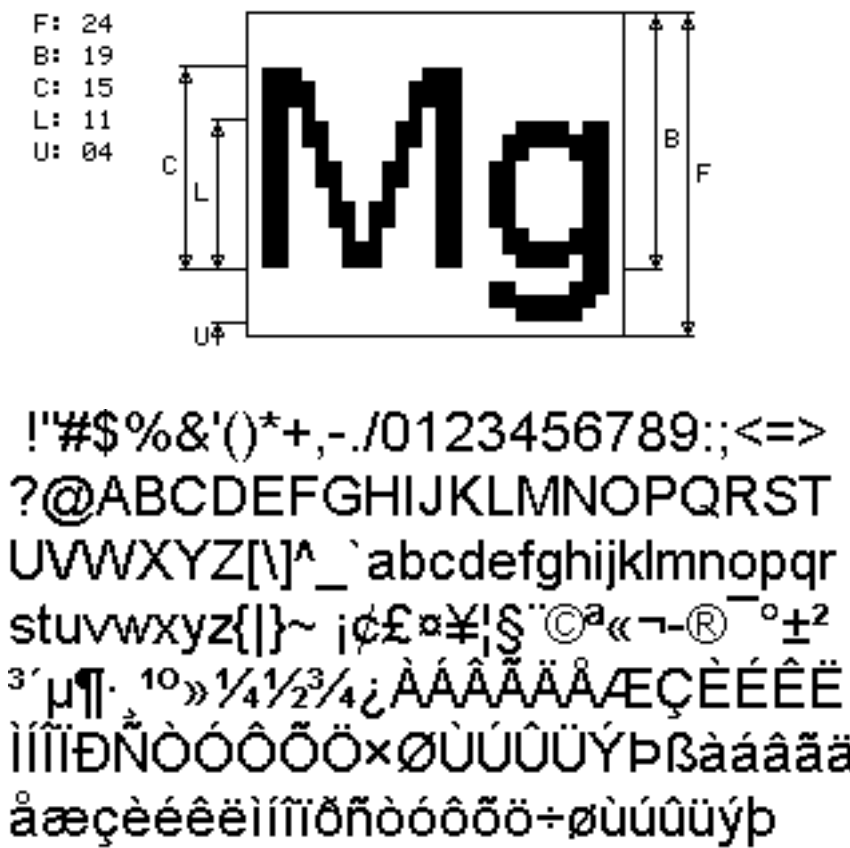
GUI\_Font24\_ASCII



!"#\$%&'()\*+,-./0123456789:;<=>  
?@ABCDEFGHIJKLMNOPQRSTUVWXYZ  
UVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~

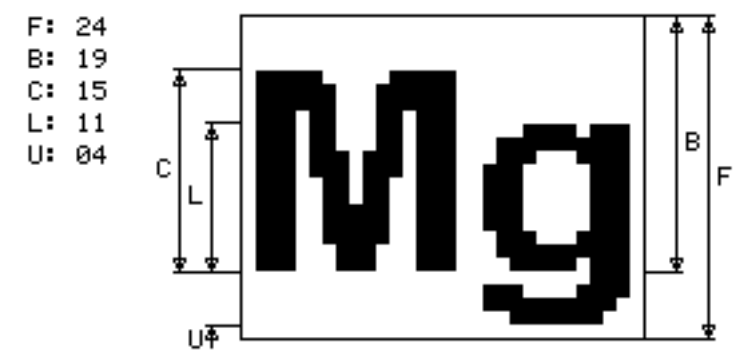
ROM Size	4786 bytes
Used files	F24_ASCII.c

# GUI\_Font24\_1



ROM Size	5022 bytes + size of(GUI_Font24_ASCII)
Used files	F24_ASCII.c, F24_1.c

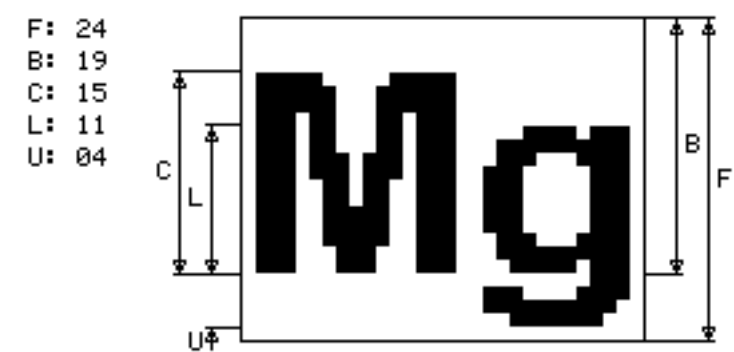
GUI\_Font24B\_ASCII



!"#\$%&'()\*+,-./0123456789:;<=>  
?@ABCDEFGHIJKLMN O PQRST  
UVWXYZ[\]^\_`abcdefghijklmnop  
qrstuvwxyz{|}~

ROM Size	4858 bytes
Used files	F24B_ASCII.c

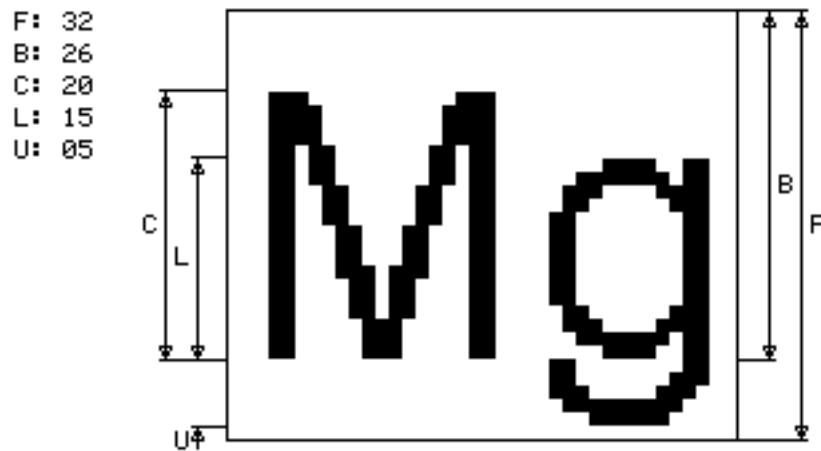
GUI\_Font24B\_1



!"#\$%&'()\*+,-./0123456789:;<=>  
 ?@ABCDEFGHIJKLMNOPQRSTUVWXYZ  
 UVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz  
 {~ ¡¢£¥¦§¨©ª«¬®¯  
 °±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈ  
 ÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßà  
 áâãäåæçèéêëìíîïðñòóôõö÷øùúû  
 üýþ

ROM Size	5022 bytes + size of (GUI_Font24B_ASCII)
Used files	F24B_ASCII.c, F24B_1.c

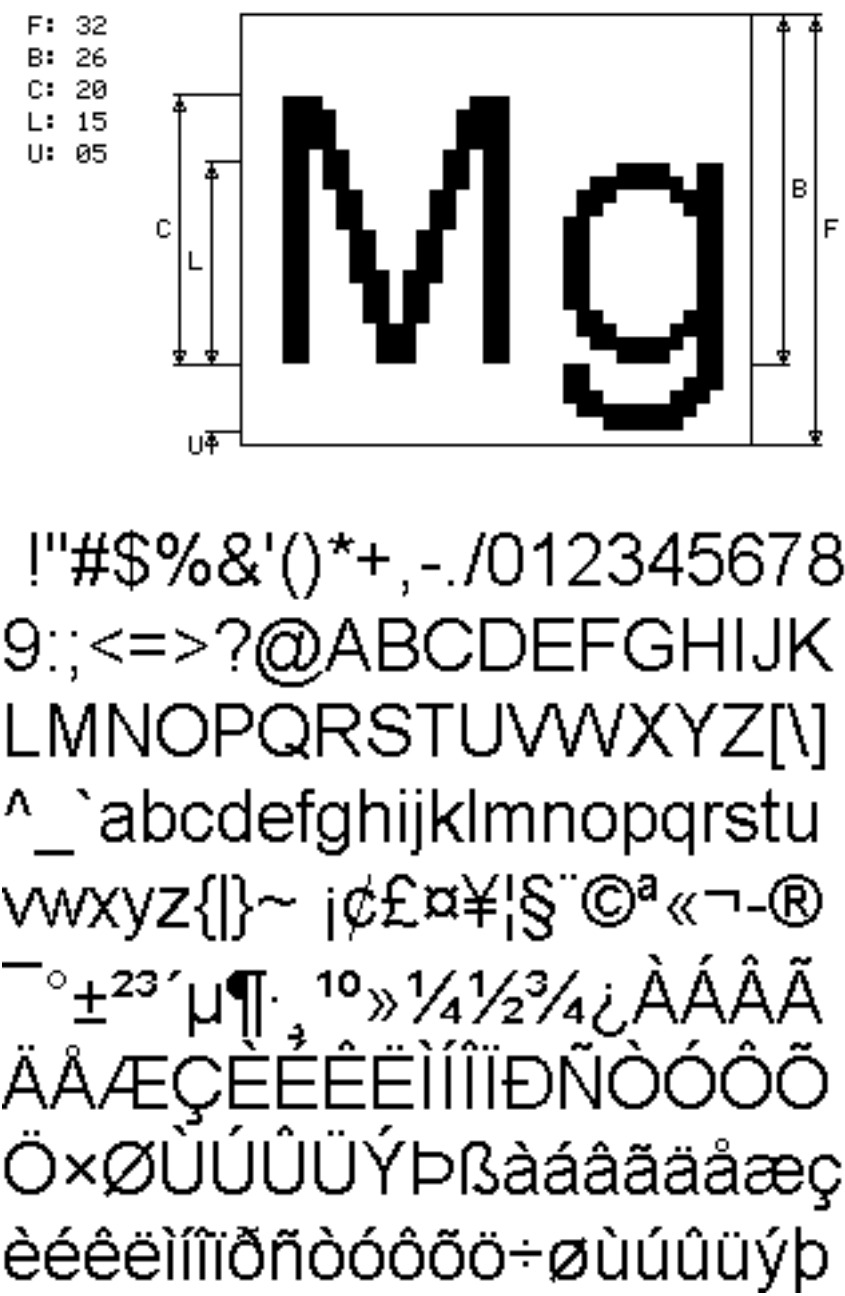
### GUI\_Font32\_ASCII



!"#\$%&'()\*+,-./012345678  
 9:;<=>?@ABCDEFGHIJK  
 LMNOPQRSTUVWXYZ[\]  
 ^\_`abcdefghijklmnopqrstu  
 vwxyz{|}~

ROM Size	7234 bytes
Used files	F32_ASCII.c

GUI\_Font32\_1



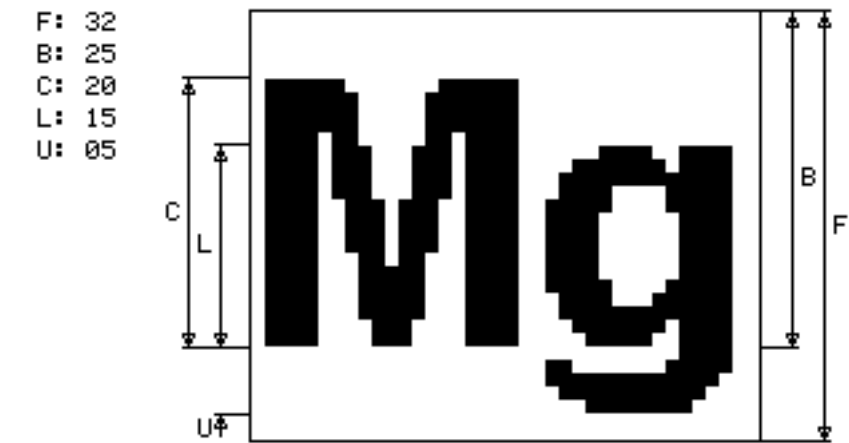
ROM Size	7734 bytes + size of (GUI_Font32_ASCII)
Used files	F32_ASCII.c, F32_1.c

GUI\_Font32B\_ASCII



ROM Size	7842 bytes
Used files	F32B_ASCII.c

GUI\_Font32B\_1



!"#\$%&'()\*+,-./01234567  
89:;<=>?@ABCDEFGHI  
JKLMNOPQRSTUVWXYZ  
Z[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~ ¡¢£¥¦§  
¨©ª«¬®¯°±²³´µ¶·¸¹º»¼½¾  
¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑ  
ÒÓÔÕÖ×ØÙÚÛÜÝÞßàá  
âãäåæçèéêëìíîïðñòóôõö  
÷øùúûüýþ

ROM Size	8118 bytes + size of (GUI_Font32B_ASCII)
Used files	F32B_ASCII.c, F32B_1.c

GUI\_FontComic24B\_ASCII

F: 24  
B: 20  
C: 17  
L: 13  
U: 04



!"#\$%&'()\*+,-./0123456789  
 :;<=>?@ABCDEFGHIJKLMNPO  
 QRSTUVWXYZ[\]^\_`abcdefgh  
 ijklmnopqrstuvwxyz{|}~□

ROM Size	6146 bytes
Used files	FComic24B_ASCII.c

### GUI\_FontComic24B\_1

F: 24  
 B: 20  
 C: 17  
 L: 13  
 U: 04



!"#\$%&'()\*+,-./0123456789  
 :;<=>?@ABCDEFGHIJKLMNPO  
 QRSTUVWXYZ[\]^\_`abcdefgh  
 ijklmnopqrstuvwxyz{|}~□ i¢£¥  
 ¦§¨©ª«¬®¯°±²³´µ¶·¸¹º»¼½  
 ¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒ  
 ÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéê  
 ëìíîïðñòóôõö÷øùúûüýþ

ROM Size	5598 bytes + size of (GUI_Font24_ASCII)
Used files	FComic24B_1c, FComic24B_ASCII.c





ROM Size	1770 bytes
Used files	F8x10_ASCII.c

GUI\_Font8x12\_ASCII


F: 12

B: 10

C: 09

L: 06

U: 02



! " # \$ % & ' ( ) \* + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F  
G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ \_ ` a b c d e f g h i j k l m  
n o p q r s t u v w x y z { | } ~ ^

ROM Size	1962 bytes
Used files	F8x12_ASCII.c

GUI\_Font8x13\_ASCII


F: 13

B: 11

C: 09

L: 06

U: 02



! " # \$ % & ' ( ) \* + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F  
G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ \_ ` a b c d e f g h i j k l m  
n o p q r s t u v w x y z { | } ~ |

ROM Size	2058 bytes
Used files	F8x13_ASCII.c

GUI\_Font8x13\_1


F: 13

B: 11

C: 09

L: 06

U: 02

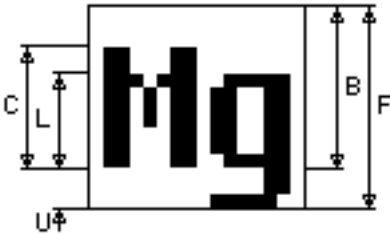


!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEF  
GHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~  
µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ

ROM Size	2070 bytes + size of (GUI_Font8x13_ASCII)
Used files	F8x13_1.c, F8x13_ASCII.c

GUI\_Font8x15B\_ASCII

F: 15  
B: 12  
C: 09  
L: 07  
U: 03

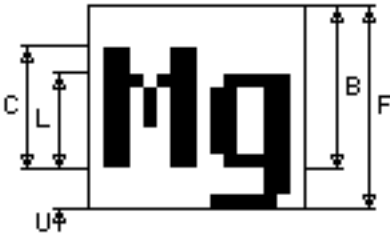


!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEF  
GHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~  
µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ

ROM Size	2250 bytes
Used files	F8x15B_ASCII.c

GUI\_Font8x15B\_1

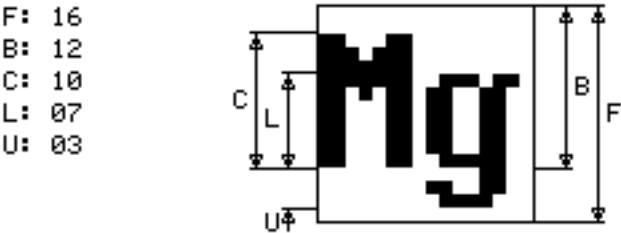
F: 15  
B: 12  
C: 09  
L: 07  
U: 03



!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEF  
GHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~  
µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ

ROM Size	2262 bytes + size of (GUI_Font8x15B_ASCII)
Used files	F8x15B_1.c, F8x15B_ASCII.c

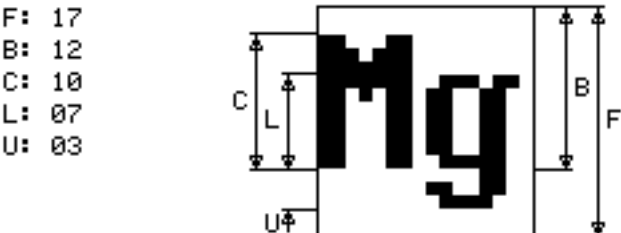
GUI\_Font8x16



!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEF  
GHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~`↔↑↓↙↘ ¡¢£¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ

ROM Size	3304 bytes
Used files	F8x16.c

GUI\_Font8x17



!"#\$%&'()\*+,-./0123456789:;<=>?@ABCDEF  
GHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~`↔↑↓↙↘ ¡¢£¥¦§¨ª«¬®¯°±²³´µ¶·¸¹º»¼½¾¿ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþ



Measurement	F:32 B:24 C:20 L:14 U:6
ROM Size	This font uses the same ROM location as GUI_Font8x16.
Used files	F8x16.c

GUI\_Font8x16x2x2

!"#\$%&'()\*+,-./012  
3456789:;<=>?@ABCDE  
FGHIJKLMNOPQRSTUVWXYZ  
[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~

Not all characters are displayed. The character set is the same as that of GUI\_Font8x16.

Measurement	F:32 B:24 C:20 L:14 U: 6
ROM Size	This font uses the same ROM location as GUI_Font8x16.
Used files	F8x16.c

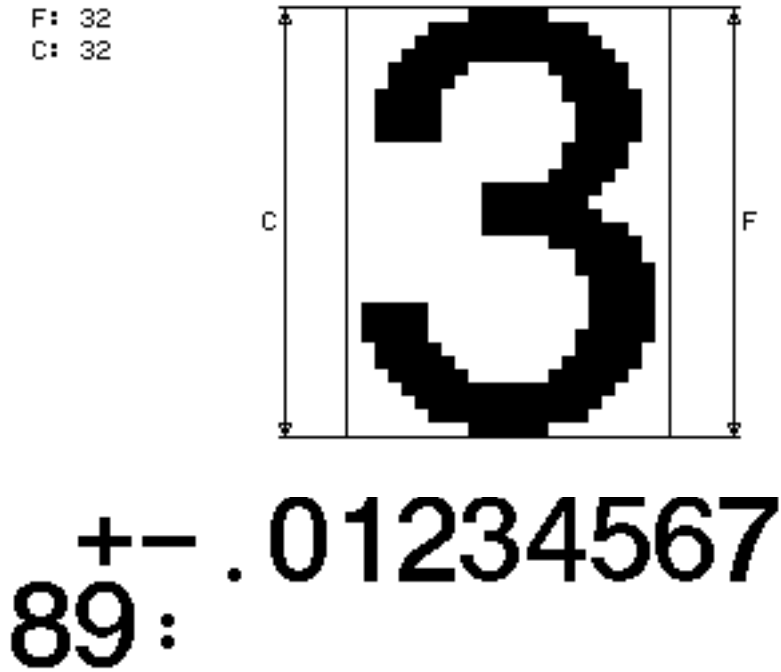
GUI\_Font8x16x3x3

!"#\$%&'()\*+,-./0123456789  
:;<=>?@ABCDEFGH  
IJKLMNOPQRS

Not all characters are displayed. The character set is the same as that of GUI\_Font8x16.

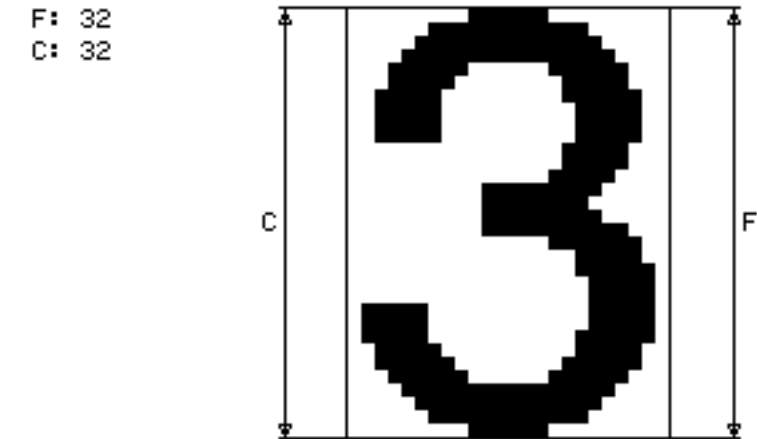
Measurement	F:48 B:36 C:30 L:21 U: 9
ROM Size	This font uses the same ROM location as GUI_Font8x16.
Used files	F8x16.c

GUI\_FontD24x32



ROM Size	1606 bytes
Used files	FD24x32.c

GUI\_FontD32



9: +-.012345678

ROM Size	1574 bytes
Used files	FD32.c

# Chapter 26

## Questions and Answers

---

Q: I use a different LCD controller. Can I still use  $\mu$ C/GUI?

A: Yes. The hardware access is done in the driver module and is completely independent of the rest of the GUI. The appropriate driver can be easily written for any controller (memory-mapped or bus-driven). Please get in touch with us.

Q: Which CPUs can I use  $\mu$ C/GUI with?

A:  $\mu$ C/GUI can be used with any CPU (or MPU) for which a C compiler exists. Of course, it will work faster on 16/32-bit CPUs than on 8-bit CPUs.

Q: Is  $\mu$ C/GUI flexible enough to do what I want to do in my application?

A:  $\mu$ C/GUI should be flexible enough for any application. If for some reason you do not think it is in your case, please contact us. Believe it or not, the source code is available.

Q: Does  $\mu$ C/GUI work in a multitask environment?

A: Yes, it has been designed with multitask kernels in mind.



# Index

---

## A

Access addresses, defining 29  
 Access routines, defining 29  
 Active window 130  
 Alias macro 14  
 Aliasing 205  
 ANSI 9, 10, 253  
 Antialiasing 195–??, 205–218  
   of fonts 205, 207  
   qualities 205, 206  
   software for 205, 237  
   with high-resolution coordinates 205, 207–208  
 API reference  
    $\mu$ C/GUI LCD 286  
   antialiasing 209  
   BUTTON 153  
   colors 100  
   EDIT 162  
   fonts 58  
   FRAMEWIN 168  
   graphics 68  
   hardware-dependent functions 265  
   LISTBOX 174  
   memory devices 113  
   MESSAGEBOX 183  
   PROGBAR 186  
   simulator 24  
   text 33  
   touch-screen 238  
   values 46  
   window manager 131  
 Application program interface (API) 10, 285  
 Arcs, drawing 84–86  
 ASCII 34, 57, 62, 65, 223  
 Auto device object 122–127

## B

Background window 130, 134  
 Banding memory device 120–122  
 Best palette option 91, 93, 94  
 Binary switch macro 14  
 Binary values, displaying 54–55  
 Bitmap converter 11, 87–97  
   command line usage 93  
   supported input formats 88  
   supported output formats 88  
   using for color conversion 90–92  
 Bitmaps 87–97

color conversion of 90–92  
 device-dependent (DDB) 88  
 device-independent (DIB) 88  
 drawing 72–74  
 full-color mode 88, 91  
 generating C files from 88, 88–90  
 manipulating 88  
 RLE compressed 89, 92, 96

Button widget 153–162  
 BUTTON\_Create 153  
 BUTTON\_Delete 154  
 BUTTON\_DisableMemdev 154  
 BUTTON\_EnableMemdev 155  
 BUTTON\_Invalidate 155  
 BUTTON\_Paint 155  
 BUTTON\_SetBitmap 156  
 BUTTON\_SetBitmapEx 156  
 BUTTON\_SetBkColor 156  
 BUTTON\_SetFont 157  
 BUTTON\_SetState 157  
 BUTTON\_SetStreamedBitmap 158  
 BUTTON\_SetText 158  
 BUTTON\_SetTextColor 158

## C

C compiler 9, 10, 19, 88  
 C files  
   as input for C compiler 88  
   converting bitmaps into 88, 88–90  
 C files, inclusion of in  $\mu$ C/GUI 17  
 C programming language 9  
 Callback mechanism 11, 132  
 Callback routines 24, 131  
   using 133–134  
 Character sets 62–65  
 Child window 135  
 Circles, drawing 81–83  
 Client area, of windows 130  
 Clip area, of windows 130  
 Clipboard 88  
 Clipping 67, 130  
 Color bar test routine 101–102  
 Color conversion, of bitmaps 88, 90–92  
 Color palettes  
   best palette option 91, 93, 94  
   custom 92–93  
   fixed 91  
 Colors 9, 99–110

- converting 99
  - logical 99
  - physical 99
  - predefined 100
- COM/SEG lines
  - configuration 247, 258–262
  - lookup table 247
- Command line usage
  - of bitmap converter 93–94
- Compile time switches 10
- Config folder 17, 20, 237
- Configuration, of  $\mu$ C/GUI 17, 30
  - for multitask access 230
  - for touch-screens 237
  - high-level 263–268
  - low-level 245–262
- Configuration, of uC/GUI
  - macro types 14
- Control characters 34, 57
- Controls (see Widgets)
- Coordinates 12, 207–208
  - high-resolution 205, 207–208
- Custom palettes
  - defining 105–106
  - file formats, for color conversion 92
  - for color conversion 92–93

## D

- Data types 13, 245
- Decimal values, displaying 46–50
- Demos 12
- Depth coordinate 130
- Device simulation 22–23, ??–24
- Device.bmp 22, 23
- Device1.bmp 22, 23
- Device-dependent bitmap (DDB) 88
- Device-independent bitmap (DIB) 88
- Directories, inclusion of 16
- Directory structure 16
  - for simulator 20
  - for Visual C++ workspace 20
- Double-byte form 224
- Drawing modes 69–70

## E

- Edit widget 162–168
- EDIT\_AddKey 165
- EDIT\_Create 163
- EDIT\_Delete 163
- EDIT\_DisableMemdev 163
- EDIT\_EnableMemdev 164
- EDIT\_GetText 165
- EDIT\_Invalidate 164
- EDIT\_Paint 164
- EDIT\_SetBkColor 165

- EDIT\_SetFont 166
- EDIT\_SetText 166
- EDIT\_SetTextColor 166
- Ellipses, drawing 83–84
- Evaluation board 11

## F

- FAQs 327
- FEDIT (see Font editor)
- Fixed color palettes 91
- Fixed palette modes 102–105
- Flickering of display 111, 147
- Floating-point calculations 67
- Floating-point values, displaying 50–54
- Font converter 11, 57, 66, 207, 219
- Font editor 65
- Font files
  - linking 57, 65
  - naming convention 300
- Fonts 11, 57–66, 299–326
  - adding 65
  - antialiased 57, 205, 207
  - available 57
  - creating additional 57
  - default 57
  - defining 11
  - file naming convention 300–301
  - included with  $\mu$ C/GUI 11, 57
  - monospaced 57, 299, 318–326
  - naming convention 299–300
  - proportional 57, 299, 301–317
  - scaling 11
  - selecting 57, 58–59
  - usage of 57
- Foreground window 130
- Frame window widget 168–174
- FRAMEWIN\_Create 168
- FRAMEWIN\_Delete 169
- FRAMEWIN\_DisableMemdev 169
- FRAMEWIN\_EnableMemdev 169
- FRAMEWIN\_Invalidate 170
- FRAMEWIN\_Paint 170
- FRAMEWIN\_SetBarColor 170
- FRAMEWIN\_SetFont 171
- FRAMEWIN\_SetText 172
- FRAMEWIN\_SetTextAlign 171
- FRAMEWIN\_SetTextColor 172
- FRAMEWIN\_SetTextPos 171
- Full bus-interface, configuration 246, 251–254
- Full-color mode, of bitmaps 88, 91
- Function replacement macro 14
- Function-level linking 17

## G

- Graphic library 9, 11, 67–86, 243

Grayscales 9, 91, 99  
 GUI configuration 264–265  
 GUI subdirectories 16, 20  
 GUI\_AA\_DisableHiRes 209  
 GUI\_AA\_DrawArc 210  
 GUI\_AA\_DrawLine 211  
 GUI\_AA\_DrawPolyOutline 211  
 GUI\_AA\_EnableHiRes 209  
 GUI\_AA\_FillCircle 212  
 GUI\_AA\_FillPolygon 212  
 GUI\_AA\_GetFactor 209  
 GUI\_AA\_SetFactor 210  
 GUI\_ALLOC\_SIZE 264  
 GUI\_AUTODEV 123  
 GUI\_AUTODEV\_INFO 124  
 GUI\_BITMAP 87  
 GUI\_Clear 43  
 GUI\_ClearRect 70  
 GUI\_Color2Index 109  
 GUI\_DEBUG\_LEVEL 264  
 GUI\_DEFAULT\_BKCOLOR 264  
 GUI\_DEFAULT\_COLOR 264  
 GUI\_DEFAULT\_FONT 264  
 GUI\_Delay 244  
 GUI\_DispBin 54  
 GUI\_DispBinAt 55  
 GUI\_DispCEOL 43  
 GUI\_DispChar 34  
 GUI\_DispCharAt 35  
 GUI\_DispChars 35  
 GUI\_DispDec 46  
 GUI\_DispDecAt 47  
 GUI\_DispDecMin 48  
 GUI\_DispDecShift 48  
 GUI\_DispDecSpace 49  
 GUI\_DispFloat 50  
 GUI\_DispFloatFix 52  
 GUI\_DispFloatMin 52  
 GUI\_DispHex 55  
 GUI\_DispHexAt 56  
 GUI\_DispSDec 49  
 GUI\_DispSDecShift 50  
 GUI\_DispSFloatFix 53  
 GUI\_DispSFloatMin 53  
 GUI\_DispString 36  
 GUI\_DispString\_UC 223  
 GUI\_DispStringAt 36  
 GUI\_DispStringAtCEOL 37  
 GUI\_DispStringInRect 37  
 GUI\_DispStringLen 38  
 GUI\_DrawArc 84  
 GUI\_DrawBitmap 72  
 GUI\_DrawBitmapExp 73  
 GUI\_DrawBitmapMag 73  
 GUI\_DrawCircle 81  
 GUI\_DrawEllipse 83  
 GUI\_DrawHLine 74  
 GUI\_DrawLine 75  
 GUI\_DrawLineRel 75  
 GUI\_DrawLineTo 75  
 GUI\_DRAWMODE\_XOR 69  
 GUI\_DrawPixel 70  
 GUI\_DrawPoint 71  
 GUI\_DrawPolygon 76  
 GUI\_DrawPolyLine 76  
 GUI\_DrawStreamedBitmap 74  
 GUI\_DrawVLine 76  
 GUI\_EnlargePolygon 77  
 GUI\_FillCircle 82  
 GUI\_FillEllipse 83  
 GUI\_FillPolygon 78  
 GUI\_FillRect 71  
 GUI\_Font10\_1 302  
 GUI\_Font10\_ASCII 302  
 GUI\_Font10S\_1 303  
 GUI\_Font10S\_ASCII 303  
 GUI\_Font13\_1 304  
 GUI\_Font13\_ASCII 304  
 GUI\_Font13H\_1 305  
 GUI\_Font13H\_ASCII 304  
 GUI\_Font13HB\_1 306  
 GUI\_Font13HB\_ASCII 305  
 GUI\_Font16\_1 307  
 GUI\_Font16\_1HK 308  
 GUI\_Font16\_ASCII 306  
 GUI\_Font16\_HK 307  
 GUI\_Font16B\_1 309  
 GUI\_Font16B\_ASCII 308  
 GUI\_Font24\_1 311  
 GUI\_Font24\_ASCII 310  
 GUI\_Font24B\_1 312  
 GUI\_Font24B\_ASCII 312  
 GUI\_Font32\_1 314  
 GUI\_Font32\_ASCII 313  
 GUI\_Font32B\_1 315  
 GUI\_Font32B\_ASCII 315  
 GUI\_Font4x6 318  
 GUI\_Font6x8 318  
 GUI\_Font6x9 318  
 GUI\_Font8\_1 302  
 GUI\_Font8\_ASCII 301  
 GUI\_Font8x10\_ASCII 319  
 GUI\_Font8x12\_ASCII 320  
 GUI\_Font8x13\_1 320  
 GUI\_Font8x13\_ASCII 320  
 GUI\_Font8x15B\_1 321  
 GUI\_Font8x15B\_ASCII 321  
 GUI\_Font8x16 322  
 GUI\_Font8x16x1x2 323  
 GUI\_Font8x16x2x2 324

GUI\_Font8x16x3x3 324  
 GUI\_Font8x17 322  
 GUI\_Font8x18 323  
 GUI\_Font8x8 319  
 GUI\_Font8x9 319  
 GUI\_FontComic18B\_1 310  
 GUI\_FontComic18B\_ASCII 309  
 GUI\_FontComic24B\_1 317  
 GUI\_FontComic24B\_ASCII 316  
 GUI\_FontD24x32 325  
 GUI\_FontD32 325  
 GUI\_GetBkColor 107  
 GUI\_GetBkColorIndex 107  
 GUI\_GetCharDistX 59  
 GUI\_GetColor 107  
 GUI\_GetColorIndex 107  
 GUI\_GetDispPosX 43  
 GUI\_GetDispPosY 43  
 GUI\_GetFont 58  
 GUI\_GetFontDistY 60  
 GUI\_GetFontInfo 60  
 GUI\_GetFontSizeY 61  
 GUI\_GetStringDistX 61  
 GUI\_GetTextAlign 40  
 GUI\_GetTime 244  
 GUI\_GETTIME() 264  
 GUI\_GetYDistOfFont 61  
 GUI\_GetYSizeOfFont 61  
 GUI\_GotoX 42  
 GUI\_GotoXY 42  
 GUI\_GotoY 42  
 GUI\_Index2Color 109  
 GUI\_Init 30  
 GUI\_InvertRect 72  
 GUI\_IsInFont 62  
 GUI\_MagnifyPolygon 78  
 GUI\_MAXTASK 230  
 GUI\_MEMDEV\_Clear 116  
 GUI\_MEMDEV\_CopyFromLCD 116  
 GUI\_MEMDEV\_CopyToLCD 115  
 GUI\_MEMDEV\_CopyToLCDAA 116  
 GUI\_MEMDEV\_Create 115  
 GUI\_MEMDEV\_CreateAuto 123  
 GUI\_MEMDEV\_Delete 115  
 GUI\_MEMDEV\_DeleteAuto 123  
 GUI\_MEMDEV\_Draw 120  
 GUI\_MEMDEV\_DrawAuto 123  
 GUI\_MEMDEV\_GetYSize 117  
 GUI\_MEMDEV\_ReduceYSize 117  
 GUI\_MEMDEV\_Select 115  
 GUI\_MEMDEV\_SetOrg 117  
 GUI\_MessageBox 183, 202  
 GUI\_OS 230  
 GUI\_RotatePolygon 79  
 GUI\_SetBkColor 108

GUI\_SetBkColorIndex 108  
 GUI\_SetColor 108  
 GUI\_SetColorIndex 108  
 GUI\_SetDrawMode 69  
 GUI\_SetFont 58  
 GUI\_SetLBorder 41  
 GUI\_SetTextAlign 41  
 GUI\_SetTextMode 40  
 GUI\_SUPPORT\_MEMDEV 264  
 GUI\_SUPPORT\_TOUCH 264  
 GUI\_SUPPORT\_UNICODE 264  
 GUI\_TEXTMODE\_NORMAL 40, 41  
 GUI\_TEXTMODE\_REVERSE 40, 41  
 GUI\_TEXTMODE\_TRANSPARENT 40, 41  
 GUI\_TEXTMODE\_XOR 40, 41  
 GUI\_TOUCH\_AD\_BOTTOM 237  
 GUI\_TOUCH\_AD\_LEFT 237  
 GUI\_TOUCH\_AD\_RIGHT 237  
 GUI\_TOUCH\_AD\_TOP 237  
 GUI\_TOUCH\_Calibrate 238  
 GUI\_TOUCH\_Exec 239  
 GUI\_TOUCH\_GetState 239  
 GUI\_TOUCH\_Init 239  
 GUI\_TOUCH\_MIRROR\_X 237  
 GUI\_TOUCH\_MIRROR\_Y 237  
 GUI\_TOUCH\_SetDefaultCalibration 240  
 GUI\_TOUCH\_SWAP\_XY 237  
 GUI\_UC\_ENDCHAR 223  
 GUI\_UC\_STARTCHAR 223  
 GUI\_WINSUPPORT 264  
 GUI\_X\_Delay 266  
 GUI\_X\_GetKey 266  
 GUI\_X\_GetTaskID 231, 268  
 GUI\_X\_GetTime 266  
 GUI\_X\_Init 266  
 GUI\_X\_Lock 231, 268  
 GUI\_X\_Log 267  
 GUI\_X\_StoreKey 267  
 GUI\_X\_Unlock 232, 267  
 GUI\_X\_WaitKey 266  
 GUIConf.h 57, 113, 230, 263, 264

## H

Handle, of a window 130  
 Hardkey simulation 23–24  
 Hello world program 30–31  
 Hexadecimal values, displaying 55–56  
 Hiding windows 130  
 High-resolution coordinates 205, 207–208

## I

I/O pins, connection to 257  
 Interrupt service routines 237  
 Invalidation, of windows 130, 132  
 ISO 8859-1 57, 63, 65

**K**

Kernel interface routines 231–233

**L****LCD**

- caching in memory 11
- configuration of 99, 245, 246–250
- connecting to microcontroller 12–13
- initialization of 30
- magnifying 246
- simulated 22
- simultaneous task access 230
- without LCD controller 13

**LCD controller**

- configuration of 245–262
- configuring additional 260–262
- connected to port/buffer 13
- initialization of 246, 250–251
- memory-mapped 13
- support for 12, 270
- with LUT hardware 110

**LCD driver 20, 269–284**

- API 285–293
- availability/selection 270
- customization of 13

**LCD\_BUSWIDTH 254****LCD\_CACHE 250****LCD\_CONTROLLER 248, 258, 270****LCD\_ENABLE\_MEM\_ACCESS 254****LCD\_ENABLE\_REG\_ACCESS 254****LCD\_FIRSTCOM 259****LCD\_FIRSTSEG 259****LCD\_GetBitsPerPixel 292****LCD\_GetFixedPalette 293****LCD\_GetNumColors 293****LCD\_GetVXSize 292****LCD\_GetVYSize 292****LCD\_GetXSize 292****LCD\_GetYSize 292****LCD\_INIT\_CONTROLLER 250, 274****LCD\_L0\_ControlCache 272, 279, 291****LCD\_L0\_DrawBitMap 288****LCD\_L0\_DrawHLine 288****LCD\_L0\_DrawPixel 288****LCD\_L0\_DrawVLine 289****LCD\_L0\_FillRect 289****LCD\_L0\_GetPixelIndex 290****LCD\_L0\_Init 287****LCD\_L0\_Off 287****LCD\_L0\_On 287****LCD\_L0\_ReInit 287****LCD\_L0\_SetLUTEntry 291****LCD\_L0\_SetPixelIndex 289****LCD\_L0\_XorPixel 290****LCD\_LASTCOM 259****LCD\_LASTSEG 259****LCD\_LUT\_COM 247****LCD\_LUT\_SEG 247****LCD\_MAX\_LOG\_COLORS 250****LCD\_MIRROR\_X 249****LCD\_MIRROR\_Y 249****LCD\_NUM\_CONTROLLERS 248****LCD\_READ\_A0 255****LCD\_READ\_A1 256****LCD\_READ\_MEM 246****LCD\_READ\_REG 252, 275****LCD\_SUPPORT\_CACHECONTROL 262****LCD\_SWAP\_RB 274****LCD\_SWAP\_XY 249****LCD\_VXSIZE 250****LCD\_VYSIZE 250****LCD\_WRITE\_A0 256****LCD\_WRITE\_A1 256****LCD\_WRITE\_MEM 246****LCD\_WRITE\_REG 253, 275****LCD\_WRITEM\_A1 257****LCD\_XMAG 246****LCD\_XORG 249****LCD\_XSIZE 248****LCD\_YMAG 246****LCD\_YORG 249****LCD\_YSIZE 248****LCD07X1 driver 271–272****LCD13XX driver 273–275****LCD159A driver 275****LCD15E05 driver 276–277****LCD15XX driver 278–279****LCD6642X driver 280****LCDConf.h 13, 14, 17, 29, 30, 105, 245****LCDEMem driver 281****LCDEMemC driver 282–283****LCDSLIn driver 284****Library, creating 17, 18****Linearization 110****Lines, drawing 74–76****List box widget 174–183****LISTBOX\_Create 174****LISTBOX\_CreateAsChild 175****LISTBOX\_DecSel 178****LISTBOX\_Delete 176****LISTBOX\_DisableMemdev 176****LISTBOX\_EnableMemdev 176****LISTBOX\_GetDefaultFont 179****LISTBOX\_GetSel 179****LISTBOX\_IncSel 178****LISTBOX\_Invalidate 177****LISTBOX\_Paint 177****LISTBOX\_SetBackColor 177****LISTBOX\_SetDefaultFont 179****LISTBOX\_SetFont 178**

LISTBOX\_SetSel 178  
 LISTBOX\_SetTextColor 178  
 Lookup table (LUT) 102, 106, 110, 285  
   for COM/SEG lines 247

## M

Memory devices 111–127  
   auto 122–127  
   banding 120–122  
   basic usage of 113  
   disabling of 113, 147  
   software for 111  
   with window manager 147  
 Memory, reducing consumption of 88, 90  
 Message box widget 183–186  
 Monospaced fonts (see Fonts)  
 Multitask environments 29, 152, 229–236

## N

NORMAL drawing mode 69  
 Normal text 38  
 Numerical value macro 14

## P

Palettes (see Color palettes)  
 Parent window 136  
 Performance 295–297  
 Pixels 12  
 Polygons, drawing 76–81  
 PROGBAR\_Create 187  
 PROGBAR\_Delete 187  
 PROGBAR\_DisableMemdev 188  
 PROGBAR\_EnableMemdev 188  
 PROGBAR\_Paint 188  
 PROGBAR\_SetBarColor 189  
 PROGBAR\_SetFont 189  
 PROGBAR\_SetMinMax 190  
 PROGBAR\_SetText 190  
 PROGBAR\_SetTextAlign 191  
 PROGBAR\_SetTextPos 191  
 PROGBAR\_SetValue 189  
 Progress bar widget 186–194  
 Proportional fonts (see Fonts)

## R

Real bus-interface 257  
 Redrawing mechanism 153  
 resource 22  
 Resource file 22, 23  
 Resource semaphore 230  
 Resource usage 295–297  
 Reverse text 39  
 RLE compression, of bitmaps 89, 92, 96  
 RTOS 229

## S

S1D13806 controller 274  
 Sample programs 12, 29  
 SED1386 controller 274  
 Selection switch macro 14  
 Shift-JIS 219–222  
   creating fonts 219  
   displaying strings 219  
 Showing windows 130  
 SIM\_HARDKEY\_GetNum 25  
 SIM\_HARDKEY\_GetState 25  
 SIM\_HARDKEY\_SetCallback 26  
 SIM\_HARDKEY\_SetMode 23, 26  
 SIM\_HARDKEY\_SetState 27  
 SIM\_SetLCDPos 24  
 SIM\_SetTransColor 23, 24  
 Simple bus-interface, configuration 247, 255–258  
 Simulator 11, 19–27, ??–27  
   directory structure for 20  
   usage of 21  
 sprintf 45  
 Subdirectories, of GUI 16  
 Syntax, conventions used 10

## T

Target systems 17  
 Text  
   alignment 40–41  
   displaying 33–38  
   modes 38–40  
   positioning 34, 42–43  
 Time-related functions 243–244  
 Toggle behavior, of hardkeys 23, 27  
 TOUCH\_X\_ActivateX 238  
 TOUCH\_X\_ActivateY 238  
 TOUCH\_X\_MeasureX 238  
 TOUCH\_X\_MeasureY 238  
 Touch-screens 11, 237–241  
   analog 237  
   configuration of 237  
   digital 237  
 Transparency 131  
   in device simulation 23  
 Transparent reversed text 39  
 Transparent text 39

## U

uC/GUI 9  
   as object version 10  
   as source version 10  
   data types used (see Data types)  
   directory structure for (see Directory structure)  
   features of 10–11  
   in multitask environments 29, 152, 229–236  
   memory requirements 297

- performance benchmark 296
- updating to newer versions 17
- uC/OS 229, 232
- Unicode 57, 65, 223–227
  - displaying characters in 223
  - displaying strings in 223
  - double-byte conversions 224
  - mixed with ASCII code 223–224

## V

- Validation, of windows 130
- Values, displaying 45–56
- Vectorized symbols 76
- Viewer 11, 21–22
- Virtual display 11
- Visual C++ 20, 21
  - directory structure for 20

## W

- Western Latin character set (see ISO 8859-1)
- Widgets 11, 151–194, 243
  - currently available 152
  - drawing 152
  - dynamic memory usage 153
  - handles of 151, 153
  - member functions of 152
  - usage of 152
- Win32 232
- Window manager 9, 11, 129–149, 151
- Window objects (see Widgets)
- Windows 129–149
  - properties of 130
  - terms associated with 130–131
- Windows, clearing 43–44
- WM\_Activate 141
- WM\_ClrHasTrans 141
- WM\_CREATE 134
- WM\_CreateWindow 134
- WM\_CreateWindowAsChild 135
- WM\_Deactivate 142
- WM\_DefaultProc 142
- WM\_DELETE 134
- WM\_DeleteWindow 136
- WM\_DisableMemdev 147
- WM\_EnableMemdev 147
- WM\_ExecIdle 134, 136, 152
- WM\_GetActiveWindow 142
- WM\_GetBackgroundWindow 143
- WM\_GetClientRect 137
- WM\_GetForegroundWindow 143
- WM\_GetHasTrans 143
- WM\_GetOrgX 137
- WM\_GetOrgY 137
- WM\_GetWindowOrgX 137
- WM\_GetWindowOrgY 137

- WM\_GetWindowRect 137
- WM\_GetWindowSizeX 138
- WM\_GetWindowSizeY 138
- WM\_HIDE 134
- WM\_HideWindow 138
- WM\_Init 134, 143
- WM\_InvalidateArea 138
- WM\_InvalidateRect 139
- WM\_InvalidateWindow 139
- WM\_MESSAGE 134
- WM\_MOVE 134
- WM\_MoveTo 139
- WM\_MoveWindow 139
- WM\_PAINT 134
- WM\_Paint 140
- WM\_ResizeWindow 140
- WM\_SelectWindow 140
- WM\_SendMessage 144
- WM\_SetBackgroundWindow 144
- WM\_SetCallback 144
- WM\_SetForegroundWindow 144
- WM\_SetHasTrans 145
- WM\_SetUserClipRect 145
- WM\_SHOW 134
- WM\_ShowWindow 141
- WM\_SIZE 134
- WM\_TOUCH 134
- WM\_USER 134
- WM\_ValidateRect 146
- WM\_ValidateWindow 146

## X

- X-axis 12, 238
- XOR drawing mode 69
- XOR text 39

## Y

- Y-axis 12, 238

## Z

- Z-position 130